Learning Board Game Rules from an Instruction Manual

Chad Mills

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science

University of Washington

2013

Committee:

Gina-Anne Levow

Fei Xia

Program Authorized to Offer Degree:

Linguistics – Computational Linguistics

University of Washington

Abstract

Learning Board Game Rules from an Instruction Manual

Chad Mills

Chair of the Supervisory Committee:
Professor Gina-Anne Levow
Department of Linguistics

Board game rulebooks offer a convenient scenario for extracting a systematic logical structure from a passage of text since the mechanisms by which board game pieces interact must be fully specified in the rulebook and outside world knowledge is irrelevant to gameplay. A representation was proposed for representing a game's rules with a tree structure of logically-connected rules, and this problem was shown to be one of a generalized class of problems in mapping text to a hierarchical, logical structure. Then a keyword-based entity- and relation-extraction system was proposed for mapping rulebook text into the corresponding logical representation, which achieved an f-measure of 11% with a high recall but very low precision, due in part to many statements in the rulebook offering strategic advice or elaboration and causing spurious rules to be proposed based on keyword matches. The keyword-based approach was compared to a machine learning approach, and the former dominated with nearly twenty times better precision at the same level of recall. This was due to the large number of rule classes to extract and the relatively small data set given this is a new problem area and all data had to be manually annotated. This provided insufficient training data the machine learning approach, which performs better on large data sets with small numbers of extraction classes. The keyword-based approach was subsequently improved with a set of domain-specific filters using context information to remove likely false positives. This improved precision over the baseline system by 296% at an expense of an 11% drop in recall, an f-measure improvement from 16% to 49% on the rule extraction subtask which is the system's bottleneck. The overall system's f-measure improved from 11% to 30%, providing a substantial improvement though leaving plenty of opportunities for future work.

1 Introduction

Humans are able to learn and execute the rules of a board game by reading the rulebook that comes with the game. The user encountering a new game may have some basic domain knowledge related to games, such as rolling dice, moving pieces along a track, choosing one of several available actions, and adding up points as they're collected or expended. Using this knowledge of basic game entities and mechanics, the rulebook fully specifies the logical structure of the game: the possible actions, their ordering, their consequences, how the game ends, etc.

Domain knowledge unrelated to game mechanics is generally not useful, and never required, to understand a game's rules. The rules must be precise to enable unambiguous interpretation among competitors with differing agendas. A board game's rulebook would never say to treat a particular entity relevant to gameplay as a boat, with the details of what impact that has on play left as an exercise for the players, leaving them to argue about how many game pieces can fit on the boat, how fast it goes, whether or not the boat was properly fueled, whether it can capsize if the passengers are all placed on one side, and any maintenance that needs to be performed on the boat. The game must unambiguously provide any relevant details in order to provide the consistency needed for players to make informed decisions and objectively determine who wins the game.

Though board games may range in domain from real estate investment (Monopoly) to war (Risk), the use of a small set of common mechanics and the irrelevancy of domain knowledge make it possible to express game rules as a combination of these common mechanics.

The approach defined in this paper translates a textual description of a board game's rules into a machine-usable representation of the rules expressed as a combination of common game mechanics. In particular, rulebooks for board games are translated into machine-readable rule representations that enable a computer to facilitate gameplay among users and enforce the rules.

2 Literature Survey

The rulebook specifies and describes the entities relevant to the corresponding game. Identifying these entities in the rulebook text builds on research in named entity recognition. Identifying the relationships

and attributes of these entities builds on the research in relation extraction. The rulebook also specifies the mechanisms and orderings of actions a player can take, which relates to research in mapping textual instructions to commands. The machine representation of the game rules builds on research in game representation systems. Furthermore, since the nature of entities in a game are fully defined by the rulebook, not depending on a player's outside knowledge of real-world counterparts, the problem also relates to natural language grounding.

An overview of the research in each of these fields follows, particularly focused on that research which most closely relates to the approach presented herein. This is meant to situate the approach presented here in its proper context, identifying research on which it depends and highlighting its distinguishing characteristics.

2.1 Named entity recognition

One essential part of translating a rulebook into game rules is identifying the entities in the rulebook that take part in actions throughout the game. This can be accomplished with techniques developed in the field of named entity recognition.

Named entity recognition is the process of identifying, within a broader span of text, entities belonging to pre-defined classes. For example, a named entity recognizer trained to find people and locations could take in a website and output all the names of people and cities on the page.

There are four main approaches to named entity recognition: hand-crafted rules, supervised learning, semi-supervised learning, and unsupervised learning. Supervised learning is the most common, with hand-crafted rules being a close second and the other two learning approaches lagging further behind (Nadeau, et. al. 2007).

Hand-crafted rule systems are built on rules which identifying text representing entities through heuristics targeted at finding symbols in the text with similarities to entities in the class being extracted (Nadeau, et. al. 2007). For example, proper names may start with capital letters, phone numbers may have digits in a particular sequence, and organizations may be represented by multiple words in upper case where the

last word is Association. The heuristics may apply to the word(s) representing the entity or to the surrounding context.

Supervised learning is the most common but requires a labeled dataset. This consists of text wherein the entities that the algorithm should be able to extract are annotated in a machine-readable form, identifying the ideal output for the named entity recognizer (Nadeau, et. al. 2007).

In one approach, a decision tree classifier was trained to identify entities based on features of a word and its surrounding context (Baluja 2000). In that system, each word in a document would be separately considered to determine whether or not it was an entity the recognizer was trained to identify. The features represent information about the word in question and other nearby words. Particular features include word-level patterns indicating case information, a dictionary lookup, part of speech tags, and some common punctuation. There were some minor modifications to this, such as treating adjacent nouns as one token, and the system performed best when only one word to each side was included as relevant context. The system performed well, with both precision and recall over 90% (Baluja 2000).

The classifier may also be a sequential classifier; instead of treating adjacent nouns as one token, the classifier may output whether the word is at the beginning, inside, or outside of a sequence of tokens that form a named entity (Ratinov et. al. 2009). This way non-noun parts of speech may be interpreted as part of an entity when combined with other parts of speech, and adjacent nouns are not always stuck being either an entity or not an entity together.

Semi-supervised learning approaches don't require a full labeled dataset like supervised learning, but they do require some labeled data and attempt to leverage those by taking into account unlabeled data (Nadeau, et. al. 2007). For example, one common approach is to start with a limited set of known entities of a given class within the labeled dataset (called the seeds), and to identify contextual patterns common across those entities. Then the patterns are applied to unstructured data in a process called bootstrapping, where new entities are identified and added to the list where they are found in text matching the previously learned patterns. Then new patterns can be found for the larger set of entities in an iterative process (Nadeau, et. al. 2007).

Unsupervised learning approaches use clustering techniques to find groups of words that appear in similar contexts (Nadeau, et. al. 2007). This is not of particular interest for this problem since our approach assumes that there are pre-defined classes of entities and relationships that are fundamental to board game systems and the goal is to map the rulebook text into this pre-defined schema, whereas unsupervised approaches generally define their own schema by identifying groupings of entities the learning algorithm identifies as similar in the data.

The literature also includes advice on how to choose among the many options available in constructing an effective named entity recognition system (Ratinov et. al. 2009). In using a sequential classifier, the BILOU system performs better than the more standard BIO; in addition to identifying the beginning, inside, and outside of an entity, the classifier should also identify tokens that are the last in an entity chunk and those that are unit-length entity chunks. And in applying an algorithm to the sequence classification problem, they found a greedy algorithm to be nearly as good as Viterbi but with a runtime nearly a hundred times better (Ratinov et. al. 2009).

There are also multiple ways to use non-local information to improve the named entity recognition system. For example, if there are four occurrences of a particular token that have all been identified as entities, the fifth occurrence of the same token is likely to be an entity as well. This non-local information has been applied three main ways: aggregating information about a token considered by a classifier and using that as a feature, applying a baseline name recognition engine and then re-running it using the output of the first engine as a feature, and using frequency information about how many times a token was tagged with a particular class from earlier in the document (Ratinov et. al. 2009). Across multiple datasets, none of these methods clearly outperformed the others. However, the value provided by each was at least partially independent. Using all three together appears to be the best approach in general (Ratinov et. al. 2009).

Outside knowledge sources may also be used to improve the recognizer. One approach uses unstructured text to cluster words together by context, forming a binary tree where nearby terms appear in similar contexts. In identifying whether or not a particular token is an entity, taking the first several steps in the path down the tree that leads to that node can form a feature that identifies more broadly

what type of contexts the term appears in over a larger corpus than the training set. Another approach to use outside knowledge is to use Wikipedia to form a massive set of dictionaries covering common names, countries, monetary units, entities, and synonyms; presence in these dictionaries can also be a feature in the entity recognition classifier. Both of these approaches work well together and provide a large improvement in performance; using them together improved the f-score of one system by ten points (Ratinov et. al. 2009). For comparison, non-local features discussed previously added nearly three points on top of that.

2.2 Relation extraction

The purpose of relation extraction is to extract relationships from text. These are commonly expressed as predicates; for example, *father-of(Manuel Blum, Avrim Blum)* is a relationship indicating that Manuel Blum is the father of Avrim Blum. In particular, this is a binary relationship since the relation has two arguments. Most research in relation extraction has focused on extracting binary relations from text, and for relations with more arguments one approach is to use a binary relation extractor followed by linking multiple binary relations together to form a larger relation (Bach et. al. 2007).

This technique is essential for use in a system extracting rules from a rulebook. Many of these rules can be expressed in the form of a relation, such as *move(pawn1,space3)* which indicates a binary relation moving entity *pawn1* to the location *space3*. Entity extraction is needed to identify the entities, and relation extraction is needed to identify the relationships between those entities.

The most common and successful relation extraction approaches use supervised learning, like in entity extraction. One approach is to take each pair of words in a sentence or nearby sentences, and then use a classifier to identify whether or not those two words form the arguments to a relation or not. To make these as effective as possible, however, there are a number of complicated features including part of speech tags, a shallow parse, and a semantic parse. These approaches perform well but the number and complicatedness of the features can make identifying the right features difficult; an alternative is to use kernel methods (Bach et. al. 2007).

Kernel methods are more suited to learning relationships between complicated data structures without the need to reduce them to a flat feature structure. For example, in a candidate relation the smallest parse subtree subsuming the nodes representing the relation arguments may be compared to subtrees of examples in the training set where the relation was and was not present. This comparison can be a mathematical similarity function that can use recursion to compare successive levels of the tree structure. This function, called a kernel function, can then be used as the discriminant function in a classifier like an SVM or voted perceptron (Bach et. al. 2007).

Kernel methods can get better scores than the feature selection based approaches using an off-the-shelf classifier, but this comes at a cost of a slower runtime (Bach et. al. 2007). The domain knowledge that would be invested into heuristics to identify good features is instead spent on identifying a good kernel function, so using kernel methods do not prevent domain knowledge from being relevant. One particular kernel approach, which uses dependency parses instead of more typical shallow parses, forms a middle ground with provably linear runtime using a dynamic algorithm but with some of the effectiveness improvements characteristic of other kernel functions (Bach et. al. 2007).

In order to identify relations with more than two arguments, a common and effective approach is to first extract binary relations from the input text and then build larger relations from this base. The entities serving as arguments in the binary relations can be treated as nodes and relations as the edges, forming an undirected graph. The edges can be weighted by classifier confidence, and the average weight of edges in a clique can without further training be used to predict whether or not all the entities in that clique form a larger relation. While there are other approaches to learn larger relations directly without depending on a binary relation extractor, since so much research has been completed on binary relation extraction this leverages all the knowledge gained through that research which has led to very effective binary relation extractors (Bach et. al. 2007).

2.3 Mapping textual instructions to commands

Branavan et. al. (2009) learns a set of goal-directed actions to be taken in a well-described environment from unstructured textual instructions. Their methodology focuses on the ability to experiment in the environment, take actions, observe changes in the environment, and estimate the effectiveness of the

actions using a manually-defined reward function. They rely on text matching between the instructions and the environment to align the actions to the instructions and use it as an input to the reward function as well. This work is applied to learning how to execute the actions in Windows support articles and to choosing which sequential moves to take in a puzzle game by following textual tutorials.

Having an environment that executes actions allows experimentation to avoid some of the annotation required in supervised environments or parallel perception and text data to identify co-occurrence patterns in for other grounded language understanding approaches. Environmental feedback and a reward function replace some of this data that may be difficult to acquire; if an environment is available, the reward function will presumably require much less effort to construct than obtaining the data needed for training a classifier.

Learning board game rules can be thought of as learning the set of valid actions rather than a single next action, but the experimentation in the environment is not possible in the board game case given that the rules governing the effect of actions are not yet established.

One alternative approach started with a corpus of text describing how to accomplish particular tasks on the Internet (Lau 2009). They hypothesized that the commands for interacting with websites are a very restricted segment of English, and they found that the vast majority of the commands in the instructions fell in the following small group: go to, click, select, and check. The data was obtained through Mechanical Turk, asking users to enumerate the steps to accomplish some goal.

They first segmented the text into segments which contained no more than one command each; this was done manually as it was not the focus of their research and represented a non-trivial problem in its own right. Then they compared three methods for extracting the commands from a text segment: a keyword-based interpreter, a grammar-based interpreter, and a machine learning-based interpreter (Lau 2009).

The keyword-based interpreter identified controls on the website that the user could interact with, and assigned them predefined keywords; for example, a hyperlink has “link,” “click,” and other terms associated with it. The text segment is compared against this using a bag-of-words approach, and the element with the most overlap is chosen as the entity to be interacted with. The type of entity

deterministically mapped to a particular action (e.g. click for a link), and if any argument was needed for that action all the keywords were removed from the text and the remainder was considered the argument (Lau 2009). So, for an instruction like entering text into a text box, words like “enter,” “text,” and “box” would be removed and the remainder of the instruction would be the text entered. This approach had a relatively high false positive rate since every text segment is assigned a command but the recall was good.

The grammar-based interpreter used a pre-defined grammar to identify each possible type of action a user could take on a website and the location of the argument where needed. This approach had a very low recall but was quite effective when it identified actions (Lau 2009). The machine learning-based interpreter using a multi-class classifier to identify each type of action based on features like the words in the instruction text. This approach was very effective at identifying whether particular text contained a command but was not particularly effective at identifying the right arguments where needed (Lau 2009). The best approach appears to be using the machine learning-based interpreter to identify which instructions contained commands, and then using the keyword-based interpreter to identify the specific action and its argument. The noisy input data with typos and other imperfections is not generally a problem faced when reading published rulebooks, but having a pre-defined schema of available actions and using a training set and classifier to identify commands bears some resemblance to extracting actions from rulebooks.

The related work in mapping text to instructions which is most closely related to the approach presented here is that of Eisenstein (Eisenstein et. al. 2009). Eisenstein attempts to learn about an environment by reading descriptive text. More specifically, they have a similar goal of learning the rules of a game by reading rulebooks. Their approach takes as given an almost-complete representation of the game system, including the game pieces, board regions, initial setup, control flow, game piece attributes, predicates with variables that can represent game pieces and attributes, and a set of glosses for each predicate which are used to align predicates to the rulebook text. They use multiple rulebooks for the game they are learning valid moves for.

Their task is to use a rulebook and the glosses corresponding to the predicates to align predicates to text and generate Boolean expressions that represent whether or not a particular move is valid. They identify relations based on word overlap between the rulebook text and a gloss, and used the dependency parser to help identify which variables to assign to which relations; for example, nearby predicates were determined to be more likely to share a variable and this was encoded in deterministic scoring functions.

Their approach is restricted to determining whether or not a particular move is valid or not, with lots of specific knowledge about the domain built into the model. They use Gibbs sampling to apply formulas to sentences, then manipulate the formulas by considering Metropolis-Hastings moves, followed by Gibbs sampling being used again to reconsider variable assignments; this procedure repeats iteratively, with one markov chain for each of the rulebooks considered. Once the procedure is completed, the formulas are ranked based on their frequency across all rulebooks. The most common ones are accepted as accurate. Relational learning is then used to group some of these Boolean expressions, taken as building blocks, into broader rules. Some smoothing is used to allow alignments where the relation matches the gloss exactly, but the vast majority of the probability mass is assigned to exact matches.

Our approach differs significantly from theirs primarily in the definition of “rules” being learned and their scope; their definition is much more restrictive than ours, limited to a Boolean function indicating whether or not a particular move is valid in a representation built around a specific game (FreeCell). Also, they use several rulebooks and utilize redundancy while for many games there is only a single, authoritative rulebook.

The reliance on only attempting to learn the rules for a single game and being able to only include actions that work for that specific game allows them to avoid dealing with the fact that most of the available game mechanics don’t apply to a particular game the rules are being learned for. All mechanics they consider for learning rules to FreeCell involve moving cards, a very small subset of possible game mechanics; both approaches, however, do take for granted the types of possible moves.

Other essential issues they do not consider include game piece state (e.g. in Checkers a piece can become a King and move backwards), broader game state (e.g. in Crazy Eights the current suit affects

what may be played), game pieces entering or leaving the board rather than being moved (e.g. getting $200 for passing Go in Monopoly), private information (players don't know the top card in a blackjack deck), randomization (shuffling a player's deck in Dominion),multiple players (e.g. Trouble), and many more. Many of these problems could be dealt with by introducing additional constants and relations, which would dramatically increase the search space, presumably lowering accuracy but still enabling a solution to the problem.

There are other problems that are not solved by a simple extension like this, though, which are centered on the method presented only providing a way to verify whether a particular move is valid and not identifying other information in the rulebook essential for understanding how to play the game. No mechanism is provided to identify when to change the state of a game piece or a global state element, among other changes that occur aside from moving a game piece from one location to another. No mechanism is provided to automatically identify what the game end conditions are or to detect when they are met. Since only one player is considered, so no control flow or game piece ownership issues are addressed. There are many other examples of this nature, but these are sufficient to demonstrate there are a range of issues not dealt with in the FreeCell case which make applying it to the generic learning of board game rules based on a common set of mechanics difficult.

2.4 Game rule system representations

The purpose of this approach is to translate a rulebook document into a logical representation of the corresponding game's rules. However, there are many options in representing a game. The representations can vary in scope (what range of games they cover?) and level of detail (how primitive or contextual are the elements of the representation language?).

Multiple formal game representation systems have been proposed (Kaiser 2007, Love et. al. 2008, Thielschr 2010). Much of the work on game representations is done in the General Game Playing research area, where researchers work to build systems capable of playing an arbitrary game effectively. One important approach is the Game Definition Language, or GDL (Love et. al. 2008). This system was first used in 2005 (Kaiser 2007) and has since been expanded to cover a broader range of games (Thielschr 2010).

GDL has only eight to ten special keywords, depending on whether or not the recent additions are included (Love et. al. 2008, Thielschr 2010). Each of these keywords represents a relation or proposition—identifying players, initializing a condition, establishing that a proposition is true, asserting that a move is legal, indicating a player has taken a move, a state representing the end of the game, a reward proposition giving a player a score for achieving a goal, and ways to set visibility and generate random numbers in limited-information and stochastic games. All the specifics are left up to the individual game to define (Thielschr 2010). This representation operates at a very low level, allowing an incredibly wide array of games to be expressed with some cost in verbosity.

The representation has no indication of game pieces, boards, or other concept typical of board games. Aside from these primitives, the language takes the form of a logic programming-like syntax (Thielschr 2010).

For example, consider the Monty Hall Game, where a car is placed behind one of three closed doors, a contestant chooses one of the doors. One of the other doors is then opened, at which point the contestant has an opportunity to switch doors or not before the remaining doors are opened. The ideal outcome for the contestant is having picked the door with the car behind it (Thielschr 2010). Symbols are defined for each of the three doors, and the proposition closed(?d) is applied to each as part of setting the initial state. The GDL specification has no notion of the proposition closed(x), but just interprets it as a proposition representing a truth about the game state at that point in time. Each game definition would have its own propositions which may or may not be named similarly.

Similarly, GDL assumes each player will choose an action each turn and all state gets updated based on logical statements defined in terms of predicates that are currently part of the game state as well as what move was chosen. No predicates are carried over from one turn to the next, so there must even be an update rule stating that any variable which is a car must also represent a car in the next turn. Updating whether or not a particular door is the current choice of the player requires a series of logical propositions combined together, including information about whether the door is currently chosen, whether the candidate switched doors this turn, and whether the candidate chooses the door. In this very simple

game, this is not too complicated to follow, but the update rules are organized around the entities in the game and not around the actions players are taking.

This works very well for planning agents where the need when a move is taken is to compute the new state from the old state and the chosen move. However, it complicates mapping rulebook text to a game representation since rulebooks generally focus on describing entities and then describing a series of actions and their consequences, organized around actions rather than all the states a particular entity may end up in. Also, with propositions just being represented by arbitrary symbols and not representing pieces or moves the system understands, two similar games could have completely different symbols for their corresponding entities and actions. This would complicate learning from mappings between text and game representations from one game to the next.

The initial version of GDL suffered from other issues that made it difficult to work with for the problem at hand. It did not allow randomization, and all state information was public knowledge (Love et. al. 2008). An alternate representation called the Regular Game Language (RGL) was created which handled these issues, as well as recognizing that games have pieces and locations (Kaiser 2007). However, while this was a significant improvement for our problem it retained many features of GDL including each player always choosing a move even on the opponents' turns and other logical contrivances that properly specify what's happening for a planning agent but aren't intuitive in the sense that they would map easily from text in a rulebook. Further, each game specifies its own propositions and relations which are represented by arbitrary symbols like in GDL, so generalizing across games is difficult in this system as well (Kaiser 2007). GDL was later updated to handle incomplete information and randomness (Thielschr 2010).

The game representations in the literature are excellent for their purpose and are capable of expressing an incredibly diverse array of games, but they present some unnecessary challenges for mapping directly from rulebook text to the game representations. Our approach is to define a higher-level representation that is less expressive but better captures cross-game generalizations, and which can be deterministically mapped to the more common GDL for interoperability.

2.5 Natural language grounding

Mooney attempts to ground language understanding in perception. He uses parallel data of simulated robot soccer games and human-provided verbal commentaries to build a system that can provide commentary on an unseen simulated game (Mooney 2008). The simulated soccer game alleviates the difficult problem of using computer vision to recognize entities and actions in real life images or videos, but retains the complexity of multiple agents and actions acting in an environment. The simulation provides information about of the state of the game, such as the locations and orientations of the simulated soccer players (Andre 2000).

Humans then provide verbal commentary on the simulated games. The system uses these two parallel data sets—the simulator state information and the verbal commentary—to pair utterances in the commentary to the events being described. The system can then recognize relevant events in a novel game simulation and provide machine-generated commentary. The long-term purpose of this work is to correlate symbols representing concepts with the real-world entities they represent, moving beyond working with symbols and toward working with complex aspects of reality with all the characteristics inherent in real objects.

The board game rule learning approach presented here differs significantly from this approach. In particular, no parallel data sets are used and all of the knowledge extracted comes from a document. However, there are some significant commonalities as well.

One interesting property of board games is that the game pieces represent entities which have properties and act in certain characteristic ways (e.g. a pawn has a color and can move from one grid cell to another). Further, the rulebook fully specifies what actions these entities are allowed to take. In this respect, similar to grounded language the ideal result is the connection of text symbols with the various properties and actions characteristic of the entities the symbols represent. Unlike in entity recognition, relation extraction, and mapping text to commands, the output is not a collection of entities, facts, or actions; instead, it's a logical system where each entity is fully described by its characteristics and actions.

The proposed approach does not fully ground these relationships in computer vision. In the soccer simulation commentary system, actions and relationships like *move*, *location*, and *owner* are understood as domain knowledge implicit in the system (Andre 2000), and the task of identifying these from images or video are left to computer vision. Similarly, the approach presented here takes for granted these same sorts of actions and relationships, including the aforementioned *move*, *location*, and *owner*.

3 Representation

First, the characteristics of games considered will be described. A schema will be proposed for representing game mechanics, followed by a representation consisting of a combination of these mechanics in a hierarchical structure to represent a game's rules. This will allow for a more formal description of the problem and provide structure needed in formulating the approach used in solving the problem.

3.1 Game Characteristics

The set of games considered is defined by the characteristics listed in Listing 1.

Listing 1 - Game Characteristics

- There is a set of players for the game, each representing an independent entity
- The focus of the gameplay is a common area visible to all players which may be divided up into regions (e.g. spaces on a Monopoly board, territories to occupy in Risk)
    - Each region may have certain properties associated with it that affect the execution logic of the game (e.g. the cost of Boardwalk in Monopoly, the neighboring countries that may be attacked in Risk)
- Each player has state information (e.g. a quantity of money, cards acquired during gameplay, supply of armies to place on the map, etc.)
    - Some of this information may be public knowledge (e.g. properties owned in Monopoly) and other information may be private (e.g. money in Power Grid)
- There are game pieces representing entities which may be controlled by a player
    - These may be located on a game region, in a common supply, or in a player's supply
    - These may have values which are additive (e.g. money, supplies) or may represent something wholly unique (e.g. the "Longest Road" card in Settlers of Catan).
- There is an initial setup, with specifications for:
    - The arrangement of game pieces (e.g. everyone starts on Go in Monopoly)
    - The assignment of an owner/controller for each of the game pieces
        - Some pieces may be part of a common supply or not controlled by anyone
    - Determining the player order
- The game has a specific control flow over time.
    - This includes the steps of a turn and what actions are legally available to be taken
        - Actions vary by game, but generally fall into often-used game mechanics (e.g. placing a piece on a game region, purchasing a game piece, etc.)
    - There is logic specifying when it is a person's turn to act, what the consequences of each action are, and when the ability to act passes to another player
- The game end conditions are specified, identifying properties that, when reached, signal the end of the game and a method for determining the winner.
- There is either a single winner or a tie between multiple players

Not all games generally regarded as board games fit these requirements (e.g. Mouse Trap), and there are some non-board-games which do meet these requirements (particularly including many card games, e.g. poker). These games are described as board games for convenience due to the large overlap between the games matching these criteria and the set of games generally regarded as board games.

3.2 Schema Terminology and Illustrative Example

A game consists of a set of game rules in a particular structure. Each game rule is an instance of a game mechanic, and each game mechanic follows a pre-defined schema common to all rules. Before

describing each of these in detail, an example will serve as an illustration to guide the reader in understanding the more generic description.

In a game, rolling a six-sided die is a rule that is an instance of the GenerateRandomNumber mechanic. Similarly, spinning a spinner or rolling a twenty-sided die would also be instances of this same mechanic.

This GenerateRandomNumber mechanic, like other rules, has a specific set of properties. It references an entity (the die), references a player (the person who rolls it), and produces an output (the number rolled). There is a fixed schema that can be used to represent a mechanic, including these properties that a mechanic may have. In addition to references and variables, one other important property is a collection of child rules that a particular mechanic may require. ConditionalAction (e.g. going up a ladder in Chutes & Ladders) requires a Condition (landing at the bottom of a ladder) to be evaluated and then has an Action (moving to the top of the ladder) that represents the effect if the condition is met.

Each game, then, is represented by a tree of rules, which are each instances of mechanics, with each mechanic fitting a pre-defined schema.

3.3 A Schema for Game Mechanics

The schema identifies the nature of game rules. A game mechanic has the properties shown in Listing 2.

Listing 2 – Properties of a Mechanic

- A name, to uniquely identify the mechanic
- A class of mechanics that it belongs to (a string that may be empty if it doesn't belong to a class)
- A variable that is produced by the execution of the mechanic and its corresponding type
- A variable that is required in order to execute the mechanic and its corresponding type
- A collection of attributes, each of which has the following properties:
    - A name, to uniquely identify the attribute within that mechanic
    - Valid types for the value of the attribute, which may be:
        - Integer
        - Boolean
        - A variable of a specific types (one a mechanic may produce or require)
        - A string of a specific category (one of a fixed list, the name of a rule, the name of a rule attribute, the value of a rule attribute)
- Whether or not a rule that's an instance of this mechanic may have additional attributes not specified by the mechanic
- A collection of children, which each:
    - Are specified as either a particular child mechanic or any mechanic belonging to a particular class
    - Have a minimum and maximum integer representing the range of how many of this child the rule may have

The schema for representing games specifies both a finite list of variable types that rules may produce or require (Entities, Location, Player, Action, Number) and a list of game mechanics fitting the above description, with all of the properties fully specified.

3.4 Game Mechanics

Now that the nature of a game mechanic has been specified, specific game mechanics will be described to illustrate the range of mechanics. This is not a comprehensive list but rather a selection intended to illustrate the nature of the schema since the full schema used to represent the games used in the experiments includes eighty-two game mechanics.

A collection of game mechanics includes everything needed to fully represent a game's rules. This includes a mechanic representing Game itself, play areas, game pieces, players, actions a player can take, control flow directing the game's progress, and identifying who wins the game. This is a wide array of information needed to represent a game, but each of these may be represented by the schema provided for game mechanics.

A game itself is represented by the Game mechanic shown in Table 1. The mechanic does not belong to a class of similar mechanics since it is a fundamental, unique representative of each individual game that can be represented using this schema. Similarly, since it does not have strong interactions with particular rules it does not produce or require variables. The only attribute it has is the name of the game.

Most of the information in this mechanic is in the children; Game requires seven specific children. Every game must have one mechanic each representing players, play areas, the set of game pieces, lookup tables, initial setup, game control, and the end of the game. The order of children is important, as will be discussed later. Every game has to have some number of players, so all the Players mechanic does is capture the minimum and maximum number of allowed players (see Table 2). Game also has Areas and GamePieces children, which are collections of the entities used in the game and the areas they're placed in. Lookup tables provide a way to provide mappings, such as the number of cards used in a game which is determined by how many players are playing. A Game also has an InitialSetup and a GameControl child, which describe the actions taken to setup and play the game, respectively. A Game's GameEnd child deals with how to determine when the game is over as well as determining who the winner is.

A Game mechanic is relatively simple, but has many child mechanics which provide much further detail about the nature of a game.

Table 1 - Mechanic: Game

Mechanic Property	Property Value
Mechanic Name	Game
Class	[none]
Variable Produced	[none]
Variable Required	[none]
Collection of Attributes	**Attribute Name** / **Attribute Type** **Game Name** / String
Additional Attributes Allowed	False
Collection of Children	**Rule or Class Name** / **Type of Child** / **Min** / **Max** **Players** / Specific / 1 / 1 **Areas** / Specific / 1 / 1 **GamePieces** / Specific / 1 / 1 **LookupTables** / Specific / 1 / 1 **InitialSetup** / Specific / 1 / 1 **GameControl** / Specific / 1 / 1 **GameEnd** / Specific / 1 / 1

Table 2 - Mechanic: Players

Mechanic Property	Property Value
Mechanic Name	Players
Class	[none]
Variable Produced	[none]
Variable Required	[none]
Collection of Attributes	**Attribute Name** / **Attribute Type** **MinNumPlayers** / Integer **MaxNumPlayers** / Integer
Additional Attributes Allowed	False
Collection of Children	[none]

The DirectedPath mechanic, shown in Table 3, belongs to the Area class indicating it's one of a number of options for game areas. Other areas may include a grid or a common supply area for money and other supplies used by a game. This illustrates how mechanics may be different from game to game.

In addition to having different attributes for various fixed mechanics common to all games, each game will also have a particular selection of mechanics within certain classes of mechanics, such as a DirectedPath being one of the areas used. The DirectedPath also has attributes indicating the length of the path and whether it's open or forms a closed loop. The children represent extra connections (e.g. for shortcuts along the path), properties of squares along the path (e.g. Boardwalk and Park Place are "dark blue" in Monopoly), and whether the square can contain many pieces or just one.

Table 3 - Mechanic: DirectedPath

Mechanic Property	Property Value
Mechanic Name	DirectedPath
Class	Area
Variable Produced	[none]
Variable Required	[none]
Collection of Attributes	**Attribute Name** / **Attribute Type** **Length** / Integer **Open** / Boolean
Additional Attributes Allowed	False
Collection of Children	**Rule or Class Name** / **Type of Child** / **Min** / **Max** **ExtraConnections** / Specific / 0 / 1 **SquareProperties** / Specific / 0 / 1 **UniquenessRequirements** / Specific / 0 / 1

The GamePiece mechanic, shown in Table 4, represents the entities manipulated in a game. The only attribute a GamePiece is required to have is a Type (e.g. pawn) but may also have additional attributes that are relevant to the rules (e.g. color).

Table 4 - Mechanic: GamePiece

Mechanic Property	Property Value
Mechanic Name	GamePiece
Class	[none]
Variable Produced	[none]
Variable Required	[none]
Collection of Attributes	**Attribute Name** **Attribute Type** **Type** String
Additional Attributes Allowed	True
Collection of Children	[none]

Choice is another mechanic, shown in Table 5, which represents a user action. Since there are many different types of actions that occur during a game and these vary from game to game, Choice belongs to an Action class. These nodes arise as children somewhere under InitialSetup or GameControl, the portions of the Game which deal with actions. The Choice action requires a Player variable, which indicates the player who makes the choice. It also requires children. One is a mechanic of the ChoiceListProvider class to provide a list of options the user may choose among as well as an ActionSequence that identifies the actions that occur as a result of the choice.

Table 5 - Mechanic: Choice

Mechanic Property	Property Value
Mechanic Name	Choice
Class	Action
Variable Produced	[none]
Variable Required	Player
Collection of Attributes	[none]
Additional Attributes Allowed	False
Collection of Children	**Rule or Class Name** / **Type of Child** / **Min** / **Max** **ChoiceListProvider** / Class / 1 / 1 **ActionSequence** / Specific / 1 / 1

If a user rolls a die, a Choice may need to be made to identify which of the player's pieces should be moved, and then the resulting action would be to move the piece a distance indicated by the die. This MovePiecesFixedDistance action is shown in Table 6.

MovePiecesFixedDistance is in the Action class, like Choice, and has one attribute for the distance moved, another for the piece(s) that are moved the fixed distance, and a flag to indicate whether the move may only be made if the piece may move the full distance. The optional children CollectionProvider and GenerateRandomNumber may be used to identify the entities moved and their distance.

Table 6 - Mechanic: MovePiecesFixedDistance

Mechanic Property	Property Value
Mechanic Name	MovePiecesFixedDistance
Class	Action
Variable Produced	[none]
Variable Required	[none]
Collection of Attributes	**Attribute Name** / **Attribute Type** **Distance** / Integer, Variable(Integer) **Entities** / Variable(Entities) **RequireFullMove** / Boolean
Additional Attributes Allowed	True
Collection of Children	**Rule or Class Name** / **Type of Child** / **Min** / **Max** **CollectionProvider** / Class / 0 / 1 **GenerateRandomNumber** / Specific / 0 / 1

An important mechanic is Turn, shown in Table 7, which represents a sequence of actions taken by a player. This is the first mechanic described which produces a variable; in this case, Turn produces a Player variable indicating whose turn it is. Its only child is an ActionSequence which allows a number of actions to be taken each turn.

Table 7 - Mechanic: Turn

Mechanic Property	Property Value
Mechanic Name	Turn
Class	[none]
Variable Produced	Player
Variable Required	[none]
Collection of Attributes	[none]
Additional Attributes Allowed	True
Collection of Children	**Rule or Class Name** / **Type of Child** / **Min** / **Max** **ActionSequence** / Specific / 1 / 1

One mechanic which may be used at the end of a game is OwnersOfPieces, shown in Table 8. This mechanic produces a Player variable, and belongs to the PlayerProvider class. This is often used to determine the winner of a game. The Pieces attribute identifies a set of pieces, which in some games may be a piece on the final space of a DirectedPath. The OwnersOfPieces mechanic identifies the owner of the piece, and thus the winner of the game in games where the owner of a piece reaching a finish line first wins the game. It can also be used in other situations, such as changing the starting player in each round to the player whose pawn is farthest behind.

Table 8 - Mechanic: OwnersOfPieces

Mechanic Property	Property Value
Mechanic Name	OwnersOfPieces
Class	PlayerProvider
Variable Produced	Player
Variable Required	[none]
Collection of Attributes	**Attribute Name** / **Attribute Type** **Pieces** / Variable(Entities)
Additional Attributes Allowed	True
Collection of Children	**Rule or Class Name** / **Type of Child** / **Min** / **Max** **CollectionProvider** / Class / 0 / 1

These examples show that mechanics, as described, can represent a wide array of game data, including the game itself, the number of players, pieces and areas used in the game, actions performed by players, and how to determine the winner of the game. The mechanics have been described, but the way they're combined into a hierarchical structure representing a single game's rules is left to the next section.

3.5 Representing a Game's Rules

Just as mechanics have a particular structure, such as a Mechanic Name, Class, Variable Produced, etc., a game's rules also have a particular structure. A game's rules are specified as a tree, with each node being an instance of a game mechanic with all of its properties fully-specified. This section elaborates on this notion, illustrating in detail how a game's rules are represented.

Each mechanic can be thought of as a template for a game rule, which is an instance of the template. A mechanic specifies the names and types of attributes a game rule modeled on that template has; the game rule itself has each of these attributes and further specifies the value of the attributes. A mechanic specifies the nature of the children: how many of each are allowed, and whether each maps to a specific mechanic or one of a class of mechanics. A game rule specifies a number of child game rules, which are

consistent in number with the type range provided by the mechanic and which are consistent with the mechanics specified (instances of a specific mechanic or instances of any mechanic of the specified class).

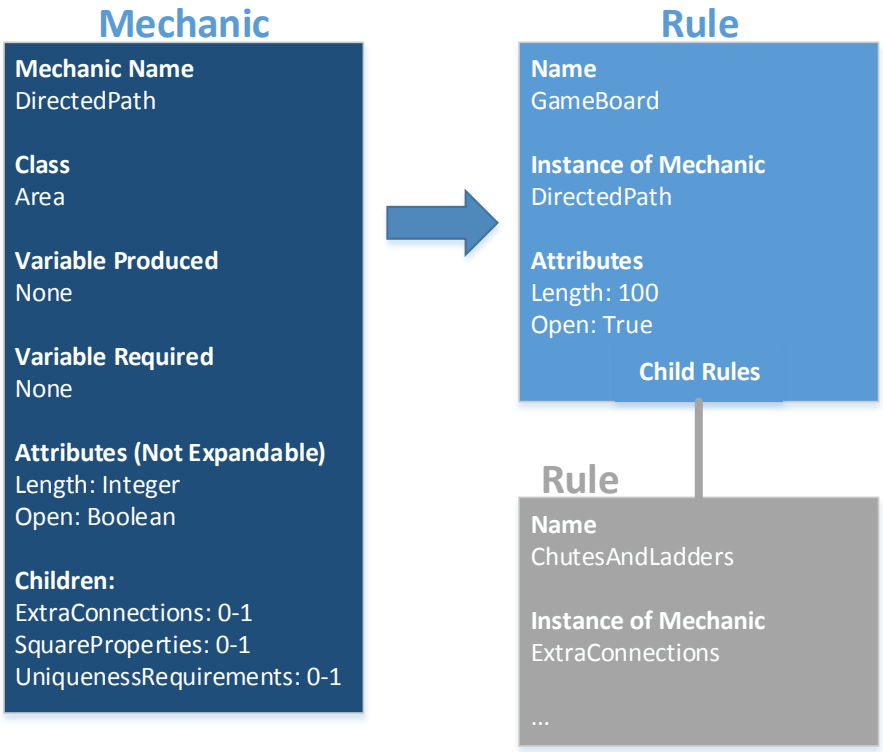
A mechanic also specifies what types of variables are produced and required for rules that are instances of that mechanic. An instance of a mechanic will further identify a mapping between variables produced and variables required between connected nodes. This is described in further detail in the next section, but each variable produced or required must be mapped to an appropriate adjacent rule.

Although all games start out with an instance of the game mechanic at their root, there are five mechanisms which allow games to be different:

- The values of the attributes of each rule are specified, which differ from game to game
- The number of children of some mechanics are variable within a range
- Children where only the class is specified may be filled by different mechanics having that class
- Variables produced and required may be mapped differently, even within a similar structure
- For mechanics which allow it, additional attributes not specified by the mechanic (and their values) may be added

An example mapping between a mechanic and a game rule is shown in Figure 1. In this example, the game board of Chutes and Ladders is shown as an instance of the DirectedPath mechanic with a length of 100, the path being open, and one child for extra connections (the chutes and ladders). The child rule is shown in part to indicate the nature of the child rule linked to and to demonstrate that it's consistent with the children specified by the DirectedPath mechanic. Like GameBoard, the instance of DirectedPath shown, the ChutesAndLadders child rule is similarly an instance of the ExtraConnections mechanic (not shown). There were no child rules corresponding to SquareProperties or UniquenessRequirements, but the mechanic allows either zero or one of each of these children so their absence is still consistent with the mechanic.

Figure 1 - Chutes and Ladders GameBoard as an Instance of DirectedPath



By convention, properties of a rule are not shown when they are empty in accordance with the mechanic the rule is an instance of. For rules which are instances of mechanics with variables produced or required, the name of any variables would be specified in the rule and where the variable produced by one rule is used by another the names would match.

As seen in Figure 1, the rule is an instance of the schema with all of the properties fully specified. The rule is given a name which is an arbitrary string that may be referenced by other rules, but a descriptive name is used to aid the reader in understanding the nature of the rule. The only way to reference a rule in this manner is in an attribute value where the type is specified in the mechanic as a string that corresponds to the name of a rule as specified in Listing 2.

The full tree of a real-world game would be very large when represented similar to the tree in Figure 1. Even most very simple games have over a hundred of rules since the rules include not just valid moves but also the entities, players, initial setup, game end conditions, etc. However, by just drawing a tree with the name of the mechanics each rule is an instance of, it is possible to see a larger section of the tree for

a game and gain a more concrete understanding of a game's structure. Even with this simplified structure the trees can be very large, but a large portion of Chutes and Ladders is shown in Figure 2.

From this tree, it is possible to see the basic structure of the Chutes and Ladders rules. Every other child of Game and its descendants are shaded differently; this does not have any significance in interpreting the rules but only serves as a visual aid to the reader in following the structure of the tree.

The structure representing actions taken throughout the game can be seen under GameControl in Figure 2. A Round node represents a series of turns that keeps repeating until some condition is met. In this case, it continues until the game is over. The Round node also has children representing the nature of the turn order (a fixed ordering) and how the starting player each round is determined (it never changes).

The most important child of Round is the Turn node, which represents what a user does on his turn. Although some of the details are omitted in this node's descendants, the basic structure of a turn is still shown: a piece is moved a fixed distance (based on a die roll) and then two conditional actions take place (the piece moves up a ladder or down a chute if the space landed on is the bottom of a ladder or the top of a chute).

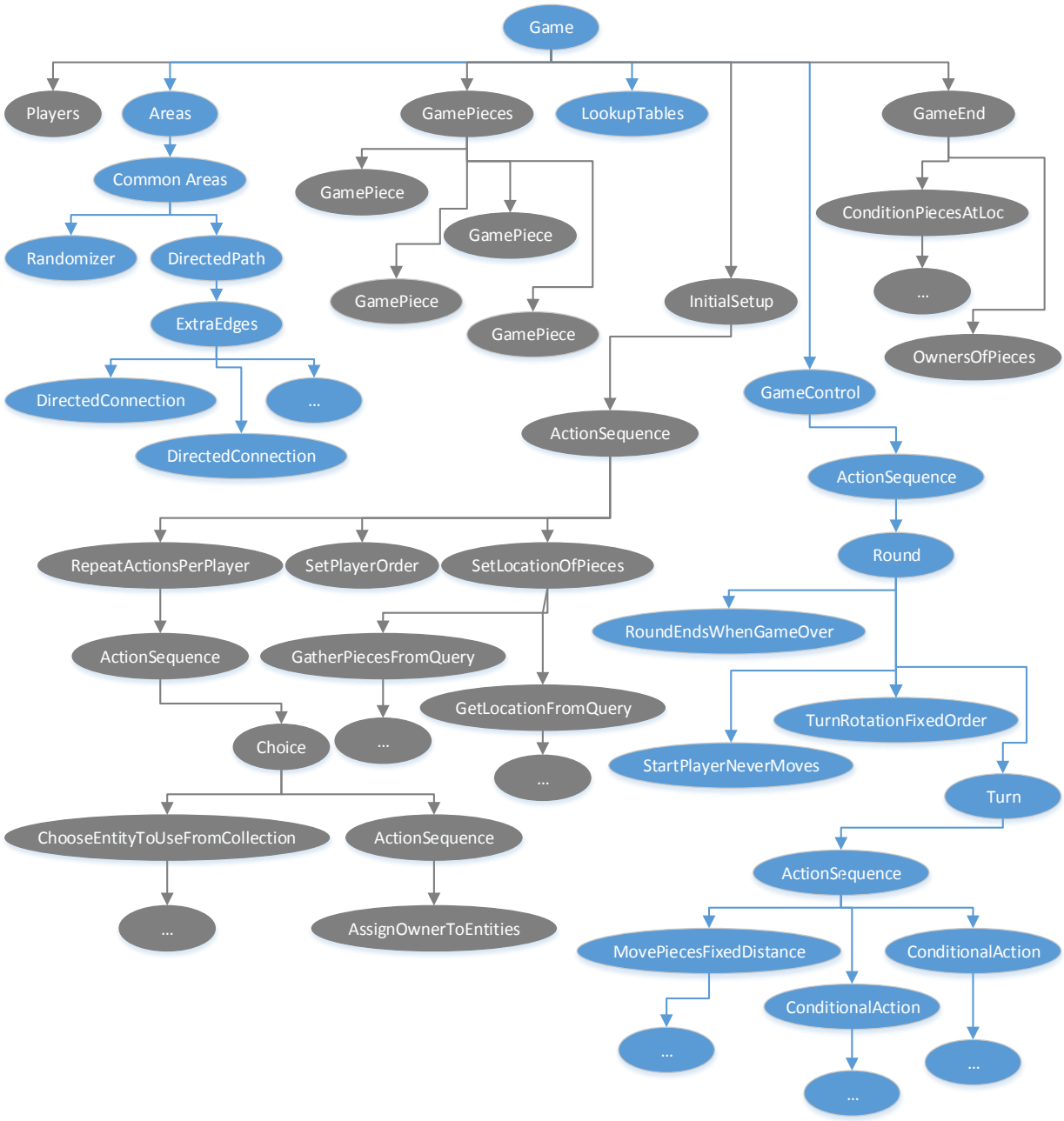


Figure 2 - Simplified Partial Game Tree for Chutes and Ladders

3.6 "Executing" a Game's Rules

From the schema described, the tree representing a game provides sufficient information to execute an instance of the game given a system that understands the nature of the game mechanics. Understanding these mechanics—how to set player order, let a user choose between a fixed set of options, etc.—forms the domain knowledge needed to play a game.

Although only a subset of a game's rules represent actions, a particular process performed on the full tree results in the game being played. Mapping rulebook text to a rule structure for a game does not require this execution, which makes this process ancillary to the scope of the present work, but if the end result of this mapping is to be a playable game there must be some straightforward way to get from the game rule representation to playing the game. To demonstrate that this works, the process is described here.

Playing the game consists of a preorder traversal of the tree of rules representing the game. In order for this to happen, the tree of rules must only include rules which are valid instances of the mechanics they represent and these must be in appropriate relationships to one another. Additionally, there must be a system which understands how to implement an instance of each of the mechanics identified. From this base, any game comprised of mechanics supported by the execution engine may be run through this preorder traversal.

In a preorder traversal of a binary tree consists in executing the left child, then the node, then the right child at each node, starting at the root. This is why the order of the children is important, which was mentioned in the previous section. Since nodes may have more than two children, left nodes for the purpose of preorder traversal are identified as those which produce variables; these must always be left of nodes which do not produce variables in a parent node's list of children. All other nodes are right nodes. The order among the left nodes and the order among the right nodes also matter; the first is executed first, then the next, and the process continues until each is executed within the group. Then, that group of nodes is considered executed and the preorder traversal may continue.

Nodes already executed may be referenced by a currently-executing node (an attribute value may refer to a node earlier in the preorder traversal order), but future nodes may not be referenced in this way. Variables produced by a node may be used in children and their descendants, as well as siblings later in the execution order of the parent's children and their descendants.

A node being executed essentially refers to the game state being altered in some way. The game state isn't anything more than the previous nodes in the traversal and any attribute values set dynamically through the execution of nodes (e.g. the location of a piece changed by the SetLocationOfPieces node).

The prior nodes may be accessed in rules by reference in an attribute value, or they may be accessed indirectly when a node representing a Query class mechanic (e.g. GatherPiecesFromQuery, GetLocationFromQuery) looks through previous nodes for entities, players, locations or areas which match specified criteria. This is how the current player's pawn is found in Chutes and Ladders in order to move it when the player moves his pawn, for example.

The execution of a node may not be deterministic. The Choice node, for example, requires presenting choices to a user and then performing some action based on what the user chose. So while all the domain knowledge is assumed to be contained within the mechanics, should be understood that some of these mechanics require player interaction and represent the user's interface with the game.

The execution of a game's nodes following the process described in this section results in the game being played.

3.7 Generalization of the Representation and Problem

The game mechanics are specified as each following a specific schema, including having a name, attributes, children, etc. The individual mechanics are treated as pieces of domain knowledge, and a game is treated as a tree comprised of instances of these mechanics.

The techniques used to build this hierarchical structure from a rulebook do so by mapping the rulebook text to instances of the provided list of mechanics and their ranges of properties. Domain knowledge behind the mechanics are not used at any point in the process, so the techniques should be common to any domain which can be represented by a set of mechanics such as this.

For instance, how-to instructions related to home repairs may be described as containing a set of tools (e.g. hammer, screwdriver), techniques (e.g. measure, cut), and hardware components (e.g. drywall, pipe). This is analogous to board games being described as containing areas, actions, and entities. A similar set of mechanics could be built to describe the domain knowledge relevant to fixing homes, and a tree representing a how-to guide could be learned from the instruction text.

Other domains such as how-to instructions are beyond the scope of this document, but it is worth noting that the techniques discussed herein are intended to be more broadly applicable. The set of interesting

problems they address is the broader set of problems that can be represented by similar tree structures regardless of the specific mechanics involved. Rather than extracting lists of entities or relationships from a document, the purpose is to construct a hierarchy of interrelated relationships and entities that adds up to one cohesive understanding of a logical system described by a document in terms of the domain knowledge (mechanics) common to such systems within the domain.

3.8 Game Characteristics, Revisited

The characteristics of games for which the rules are being learned were specified in Listing 1. Virtually all of these characteristics are restricted not by the representation but the particular game mechanics used to represent the domain knowledge used for this system. All eight of the primary requirements correspond directly to mechanics that are always children or grandchildren of the root Game rule. Thus, the restrictions enumerated are domain restrictions required by the portion of the representation most associated with domain knowledge: the mechanics.

The tree structure used is very flexible and can represent a broad array of logical systems. The specific mechanics used, though, delimit the domain to form the language used by all such logical systems in the class using that set of mechanics.

Expanding or reducing the number of games included would entail not changing the processes to map rulebook text to rules, but rather changing the mechanics that text in the rulebook is mapped to. In this sense the representation is much more expressive than needed for the game domain and the particular domain knowledge needed for the game domain is encoded in the mechanics used rather than the representation itself.

4 Problem Statement

The problem addressed herein is taking the text from the rulebook of a game satisfying the criteria of eligibility listed in Listing 1 and automatically constructing a tree representing the game's rules as described in Section 3.5.

This treats as given a set of game mechanics capable of representing these rules and following the schema described in Section 3.3, as well as a list of variable types described in the same section.

One further restriction is also assumed at this point. While the vast majority of rules are described in the rulebook, some games do not fully describe the game pieces and board, assuming the players will understand by looking at these pieces rather than by reading about them in the rulebook. Other games describe the contents of the box, including any relevant attributes and sufficient information to identify the mechanics they represent without needing to look at the physical objects. A full solution to this problem would require computer vision and related disciplines, so to keep the problem tractable any rules not fully specified in the rulebook will be manually annotated instead of learned automatically.

5 Methodology

The problem of mapping rulebook text to a tree representing the game rules can be decomposed into a number of sub-problems including: identifying each rule mechanic, mapping parent-child relationships, ordering the children of a parent, identifying attributes, identifying attribute values, mapping attributes to their values, and mapping attributes to rules.

Identifying rule mechanics, identifying attributes, and identifying attribute values can be modeled as entity extraction problems, while mapping parent-child relationships, mapping attributes to their values, and mapping attributes to rules can be modeled as relation extraction problems. Ordering the children of a parent is the only sub-problem not put into one of these two categories, and that is handled separately through heuristics.

When decomposing the problem, it's necessary to perform some subtasks before others, and this is illustrated in Figure 3. This includes one oversimplification that is addressed later, but illustrates the basic structure of the subtasks and how some of them are grouped together into tasks of similar structure (entity extraction, relation extraction). The entity extraction tasks must be performed before related relation extraction tasks that reference those entities, and finally the ordering of children can only happen once parent-child relationships are formed. This creates a dependency graph indicating which subtasks depend on which other subtasks being completed first.

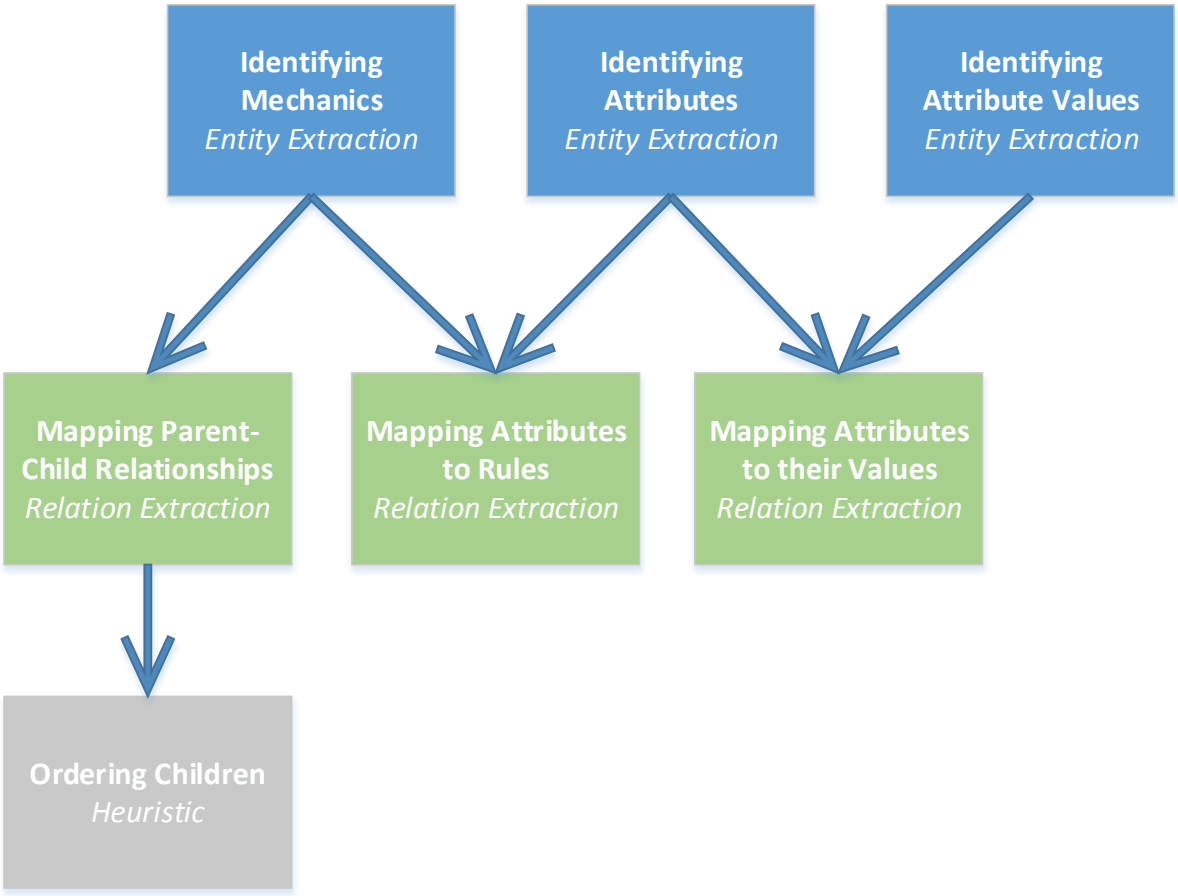


Figure 3 - Relationship between task subcomponents

5.1 Basic Approach for Entity and Relation Extraction

The data used for evaluation is discussed in Section 5.7.1, but one aspect of the data had a significant impact on the basic approach used so it needs to be noted here. Since this is not a well-established problems with available corpora for training or evaluation, the data needed to be manually annotated for this study which significantly diminished the volume available relative to many well-studied entity and relation extraction tasks.

Given the small volume of data, a keyword-based approach was used for entity extraction. Each possible entity that could be extracted conforms to a game mechanic or property thereof, so keywords were associated with these entities that represent words that might be used to represent them in text. A machine learning based approach to entity extraction was also taken for comparison to validate that state of the art techniques on larger data sets would not translate well to this small data set.

The relation extraction subtasks involve identifying binary relationships among these extracted entities (rule-rule relations for identifying parent-child relationships, attribute-rule relations for mapping attributes to rules, and attribute-value relations for mapping attributes to their values).

5.2 Entity Extraction

The primary approach to entity extraction is a keyword-based approach detailed in 5.2.1. The machine learning-based approach used for comparison purposes is described in 5.2.2.

5.2.1 Keyword-based Entity Extraction

The mechanics from Section 3.3 include all the information needed to form a rule (an instance of a mechanic) except the attributes and children. Children can only be other rules, and the list of attributes are also specified as well as the type of their values. Treated as three distinct types of entities, mechanics, attributes, and attribute values form all the basic information to be extracted from the rulebook text. A rule complex, not any one of these entities but the combination of all of them plus the relations between them, so rules are not directly being extracted. Rather, rules are being formed through this process of entity extraction followed by relation extraction.

The full list of possible mechanics is considered given, known in advance and useable for extraction. Since each mechanic has a list of attributes, the union of attributes across all these mechanics can be used to compile a list of all possible attributes. The attributes have a type specified, which restricts their values to specified domains (e.g. integers) and an assumption can be made that these values fall within a common range found in board games so a list of possible values can be compiled from this as well.

Given each mechanic, attribute, and attribute value that can be extracted from the rulebook text, keywords are generated to represent possible ways that entity may be referenced in text. For example, the mechanic Choice is represented by the keywords “choose”, “pick”, and “select”, while the mechanic ConditionalAction is represented by the keywords “if” and “when”. For attributes, “square” or “space” are keywords used to represent the presence of a location attribute. Attribute values are represented by strings representing numbers, specific strings the attribute values may take, etc.

By compiling a list of keywords for each entity that can be extracted, when the keyword is found in the rule text that entity is proposed as a candidate mechanic, attribute, or attribute value. Some of these may be filtered out later, as described in Section 5.5, but the initial entity extraction proposes each entity where the keyword exists in the rule's text. This same approach is used for all three categories of entity extraction used.

In some cases, the same keyword may represent both an attribute and its value. So, the same span of text may indicate both the nature of the attribute being described and its value. Alternatively, sometimes the attribute will be indicated with one span of text while its value is indicated by a different span of text. Since each extraction is treated independently, all matching keywords are proposed as extracted entities for each entity the keyword matches. As a result, this does not pose a problem.

One specific example from Chutes and Ladders, which is dense with keywords, is shown in Figure 4 to illustrate the types of entities extracted and keywords they map to. The coloring is not significant; every other identified word is in an alternate color to identify the spans of text involved and avoid ambiguity around two words being part of the same span or separate spans.

In this example, examples are included for extracting mechanics, attributes, and attribute values. The mechanics extracted include actions (Turn, Randomize, MovePiecesFixedDistance) as well as entities (GamePiece, Randomizer).

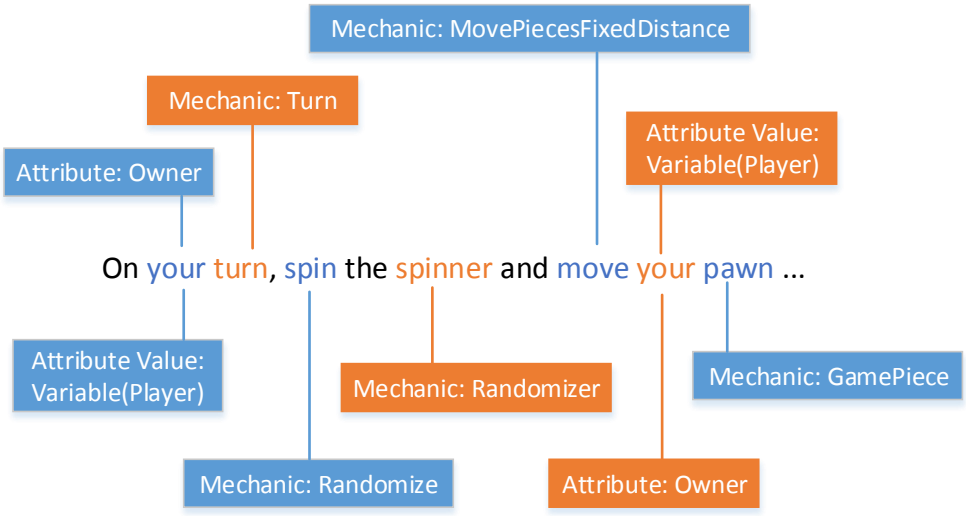


Figure 4 - Extraction Example from Chutes and Ladders

There is one minor complication that does not allow all of the entities to be extracted at the same time. One possible value for an attribute is the name of a specific rule, and rules are identified separately (by mechanic). Similarly, attribute values may also be the names of other extracted attributes; while these can be specifically enumerated for any given set of rules extracted, and the possible schemas for these references are known in advance, the only way to reference a rule or attribute instance is by having extracted it first. This could also be accomplished by identifying a candidate reference to a mechanic and linking the specific instances through a later relation extraction step, but by waiting for the rules and attributes to be extracted this has a side benefit of only allowing references to rules that were found and reducing the number of extraneous candidate references identified. As a result, the process is slightly more complicated than Figure 4 depicted and the full version is shown in Figure 5.

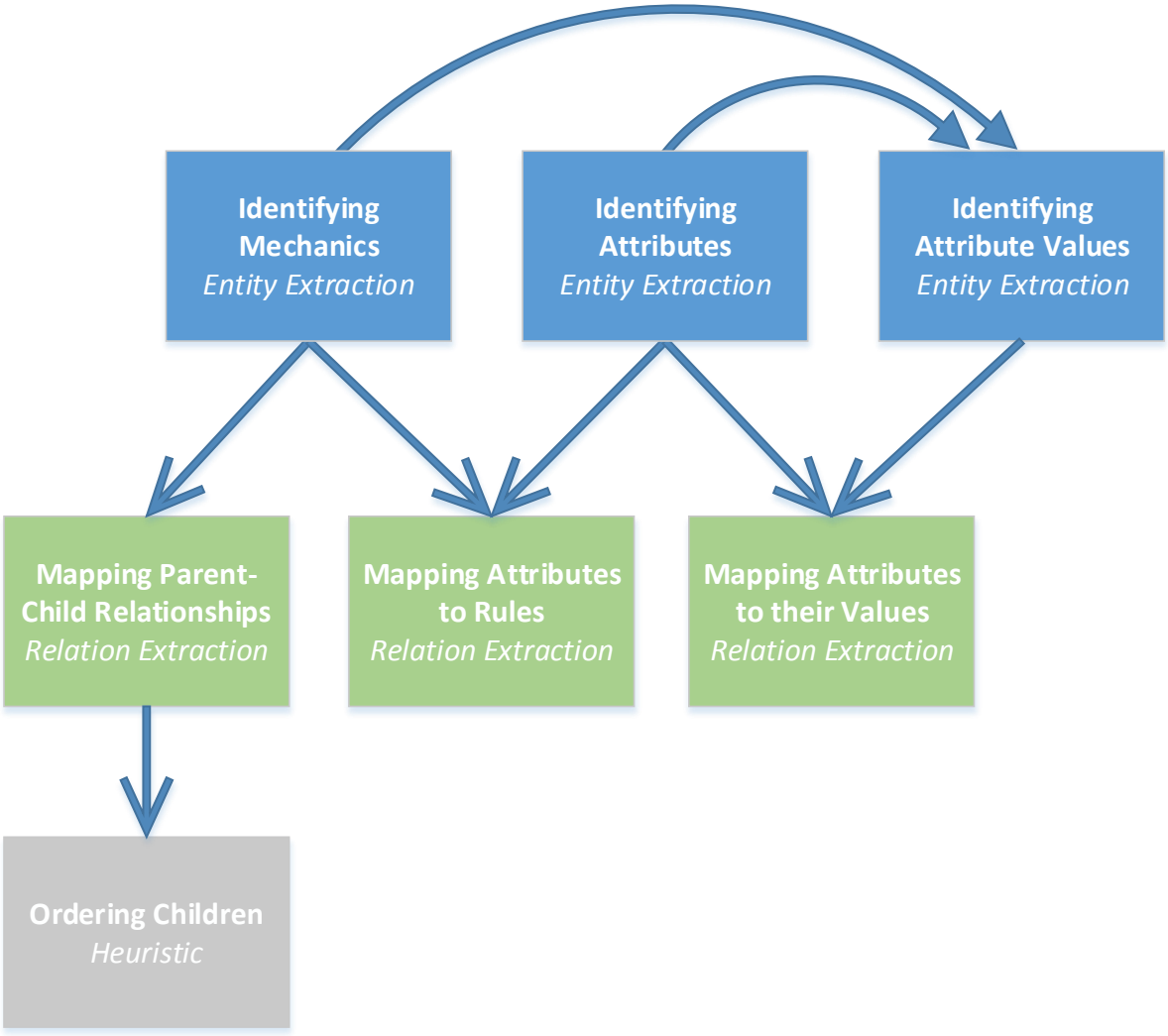


Figure 5 - Task Subcomponents Revisited: Attribute Values may be Rule Names

5.2.2 Machine Learning-based Entity Extraction

While a machine learning-based approach to entity extraction is the state of the art in many entity classes, the documentation for widely-available training systems suggested that orders of magnitude more data would be needed to effectively train entity extractors for each rule class. Due to the large abundance of classes and small amount of labeled data available for training, a machine learning based

approach was not expected to perform as well as a keyword-based approach. Nonetheless, given this approach's success in many cases, a system following this approach was created to test this assumption.

Each token in a rulebook was treated as a candidate token that could be any of the entities being extracted, and features were extracted for a maximum entropy classifier with one record for each token. The features were based on the token itself, the parse tree from the Stanford parser, and similar features for nearby words.

The specific features used include the token itself, the lemma associated with that token, its part of speech according to the Stanford parser, any named entities associated with the token given the fixed set of entity classes parsed by the Stanford Named Entity Recognizer, the token's location in the sentence, and n-grams of the parse tree. These same features were extracted for the other tokens in the same sentence, as well as features identifying the distance between these nearby words and the one being considered as a candidate entity.

Since there are many classes of entities, such as each rule, and since nothing in the schema requires that one token map to no more than one rule, a separate classifier was used for each entity that could be extracted rather than having one multi-class classifier.

5.3 Relation Extraction

Relation extraction gets the benefit of starting with a list of proposed entities that relations can be formed between as well as a schema identifying valid and invalid relationships and information about where in the rule document each candidate entity was found. Each of the three types of relation extraction follow this same pattern, and what differs is only the types of entities being related (rule-rule relations, rule-attribute relations, and attribute-value relations).

5.3.1 Basic Approach

Since the entities potentially involved in relations are already extracted, since there's a schema identifying which relationships are possible, and since the relations don't involve any outside entities, the approach to relation extraction used involves mapping extracted entities together based on the schema and text alignment of the entities extracted.

This process does not involve looking at the text for additional keywords indicating a relationship, but rather maps already-extracted entities, attributes and values into relations that fit the existing schema. There would be many possible relations considering all possible valid combinations of extracted entities in the rule text, so the distance between the keywords representing each entity is used to limit proposed relations to a reasonable subset. Heuristics are used to limit the large set of possible pairs to a smaller set that attempts to keep the relations that exist in the text while filtering out the invalid relations that aren't really represented by the rule text.

5.3.2 Mapping Parent-Child Relationships

To identify parent-child relationships, the schema defined by the mechanic corresponding to each proposed rule indicates an allowed set of children along with the quantity of each child allowed. All of the other proposed rules which could be valid children are set aside as candidates.

For a given proposed rule, when all of the children matching one specific child schema are found, the distance between the keywords of the proposed rule and the candidate child is measured in the number of sentences apart they are. The minimum distance is taken, and all rules at that minimum distance are considered; from here, as many rules as allowed by the schema are taken from this subset, starting with the closest by word distance and moving toward the furthest away within this subset. This filters the set of proposed rules to the closest allowed.

For example, from the entity extraction example in Figure 4, the *MovePiecesFixedDistance* rule requires a set of pieces to move and a distance to move them; it has two children, one for each of these, according to its schema. There are a number of possible rules which can be children to fulfill each of these roles. For the distance being moved, for example, this could be filled by a number directly extracted from the text, the *TableLookup* rule, or a *Randomize* rule, among others.

As can be seen in the figure, there's a *Randomize* rule proposed earlier in the sentence, so this is proposed as the child and serves to provide the number that represents the distance the piece will move. This *Randomize* also requires a child, which is fulfilled by the nearby *Randomizer* mechanic

corresponding to the nearby word “spinner”. In both cases, the correct child is the nearest mechanic proposed as a rule based on its own keyword being found nearby.

Each level of the hierarchy represents a rule that is proposed due to a keyword match, so all of these parent-child relationships are proposed in a single pass since all possible parents and children are mapped to the nearest possible child before any filtering takes place. It is not iteratively built up in separate passes.

5.3.3 Mapping Attributes to Rules

For each proposed rule, all the attributes that match the schema’s attributes are considered as potential relations. Unlike parent-child mappings, each rule will have exactly one of each attribute in the rule’s schema. This results in a slightly simpler set of heuristics due to no need to accommodate multiple mappings at a similar distance.

For each proposed rule-attribute mapping, the word distance is calculated between the keywords of the rule and the attribute in the rule text. The mapping with the shortest distance is chosen, and if there are mappings in each direction at the same distance one of them is chosen randomly.

5.3.4 Mapping Attributes to their Values

For each attribute, the attribute values with a matching type are possible values that can be associated with that attribute. From these potential relations, that with the shortest distance (measured in words) between keywords representing the rule and attribute is chosen as the relation to propose to map an attribute to its value. This works the same as the way attributes are mapped to rules since attributes only have one value just as rules only have one of each attribute indicated by the schema.

5.4 Ordering Children

After extracting the rules and their parent-child relations, some rules will have multiple children and the order is significant. The simple heuristic used here is to order the children based on the same order the keywords representing the child rules appear in the rulebook. However, if there are variables produced by a rule the rule must come before rules which don’t produce variables; among rules which produce and require variables, they need to go in the middle between those rules which produce and consume

variables. After this split is made into these three sections, the rules are ordered by their order in the rulebook. Finally, if any rule in the middle section requires a variable of a type not produced by preceding rules and a later rule in the same section produces that rule, the rule is moved after the rule producing the needed variable type.

5.5 Purging Entities

The output of this process is ideally a tree of rules which perfectly represents the rules of the game and their relation to one another, with each attribute and attribute value correctly identified. However, errors in the process will result in some rules being proposed which are not really part of the rules (false positives), and others which are part of the rules but which are not proposed (false negatives). Using a keyword-based approach, the system is promiscuous in identifying many proposed entities which may not really have a counterpart in the rules.

One way to reduce this problem is to filter out entities which are proposed but don't end up connected to other entities by relations. This is a post-processing step after all the entities and relationships have been identified and the children have been ordered.

Rules without either a parent or child in the same sentence are filtered out, as well as attributes that aren't mapped to a rule and attribute values that aren't mapped to an attribute. This keeps rules, attributes and values which are connected to other rules in nearby portions of the document and filters out rules that were likely generated too aggressively.

5.6 Additional Rules Attempted – Improving over the Baseline

In addition to this baseline approach described in sections 5.1-5.5, additional rules were implemented in an attempt to improve the system. These were based on observations of the system's performance and mistakes it made on one of the games analyzed, Chutes and Ladders, so as to gain insight that would lead to improvements without tainting the other three games as test data. As will be discussed in section 6, the main limitation of this baseline system was a very low precision due to keywords often appearing as examples or elaborations without introducing new rules. Therefore, many of the additional rules pertain to filtering out the extraneously identified rules.

Note that in Section 3.7 the problem was generalized such that it could represent a wide variety of problems with the same basic structure as the problem of learning game rules from an instruction manual, such as constructing furniture from how-to manuals. This led to a clarification to the game-specific task in Section 3.8 into a form suitable for application to these sister domains. The filters applied here do not fit within this generalized problem statement.

All of the domain knowledge is supposed to be encoded in the keywords and the allowed rules, as well as their parent, child, attribute, and attribute-value relationships. By encoding additional domain knowledge in a set of keyword-based filters, this constraint is no longer being adhered to. It is entirely possible some of these filters may generalize to other domains as well, but even optimistically the keywords associated with the filters would likely have some element of domain-specificity. Others, such as such as filtering out rules from sentences with only one keyword (Section 5.6.8) or any evaluative words (Section 5.6.12), seem unlikely to be apply across a broad range of domains as they pertain to the empirical density of keywords in rulebooks as well as the fact that rulebooks in particular offer strategic hints to players often marked by evaluative words which can safely be ignored.

The filters applied in Section 5.6.1 through Section 5.6.15 were implemented as a layer after the entity extraction and before relation extraction and evaluation, all described earlier in Section 5. Due to the keyword-based approach generating large numbers of rules and leading to a low precision, as will be discussed in Section 6, these filters were an attempt take additional context information into consideration and filter out some of the extraneously-identified entities and relations. The rules use context information at the keyword, phrase, or sentence level, consisting of the keyword itself, other tokens in the sentence, parts of speech of the keyword, parts of speech of other tokens in the sentence, or the dependency information from the parse tree. With this context information, rules were crafted to filter out extraneously-identified entities, attributes, or relations.

The filters' place in the system architecture can be seen in Figure 6, a revised version of Figure 5 that shows the way filters are applied after entity extraction but before relation extraction.

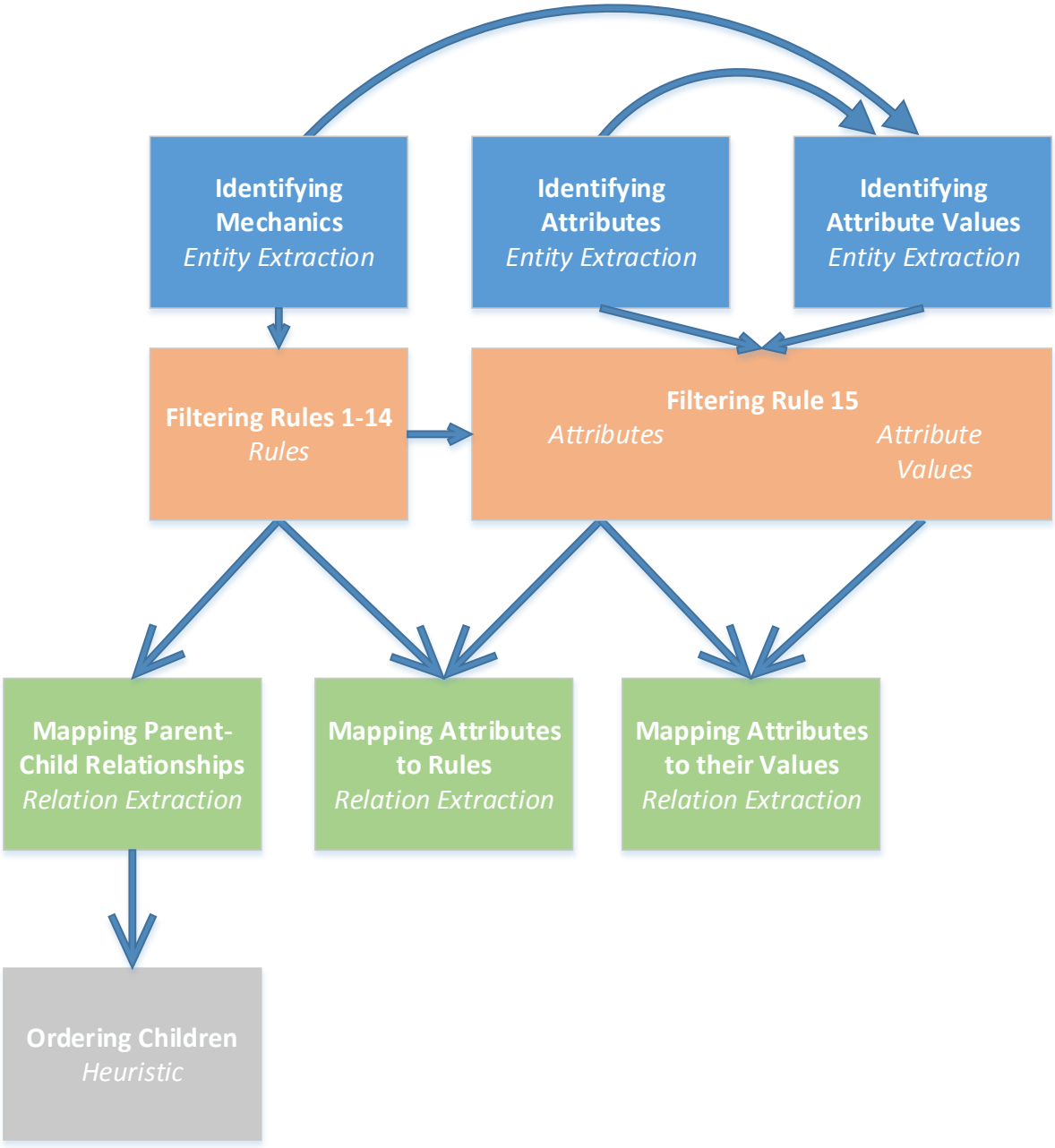


Figure 6 - Filters and Their Place in the System

This also shows how most filters are simply filters of game mechanics or rules, while these filters feed into the last filter which applies to the other entity extraction tasks, attribute and attribute value detection. The

main focus was on filtering the game mechanics and the final rule cashed in on the gains obtained through this process by further filtering out attributes and values that were likely to be associated with the game mechanics filtered in this way.

The remainder of this section will discuss each of the filters in turn, describing the context information taken into account and how this was used to filter out entities that had already been extracted but were likely to be false positives.

5.6.1 Filtering Header Sentences

Rulebooks contain headings, which often include keywords relevant to the essential entities and actions described by the game's rules. As a result, it is relatively common for a keyword for a rule to appear in a heading. However, headings do not introduce rules and these are generally false positives. Exacerbating this situation, the parser used seems to parse the heading as well as the sentence following it as one sentence, making identifying the heading and filtering out rules without attachments within the same sentence both more difficult.

Headers were identified as parse phrases with either all words capitalized or at least four words capitalized. This rule filtered out all proposed rules in headings identified this way.

5.6.2 Filtering Sentences with Too Many Rules

Some sentences, as separated by the parser, contained an excessive number of rules. Due to the broad definition of rules used in this task, including entities, actions, and relations, some sentences did have several related rules. However, some sentences had many more than would reasonably exist. This filter removed all rules proposed in a sentence with an excessive number of proposed rules.

5.6.3 Filter Ineffective Keywords

Keywords were originally found for rules by looking through rulebooks of other games, performing web searches, and identifying synonyms of other proposed keywords. Many keywords proposed to represent particular rules were not actually finding rules and some of these were generating false positives. While broader sets of keywords were desired to achieve a reasonable recall on an unknown test set, the particularly egregious keywords were filtered out.

5.6.4 Filter Conditional Sentences

One issue that makes extracting rules from a rulebook difficult is that the text of rulebooks often elaborates on rules, offering advice to players or providing an example of a just-introduced rule. Unfortunately, these are easy to identify as rules themselves since they use many of the same words. Sometimes, though, the rulebook offers enough clues that some of these extraneous words can be filtered out. In particular, many mistakes were observed in sentences with certain conditional words, including “should”, “might”, and other similar words. This filter attempted to remove any rules identified in sentences that contained these conditional keywords. Some other conditional words, like “if”, were present in many actual rules being introduced, so this is not an exhaustive filtering of conditional words and focused instead on speculative or instructive conditionals.

5.6.5 Filter Early Sentences

Most rulebooks start with an overview of the game, which provides some context to readers but doesn’t introduce rules. This filter excluded sentences from the first section of the rulebook (if small) or the first several sentences (if not separated out in a small section).

5.6.6 Filter Sentences without Both Verb and Noun Keywords

Since rules often involve actions on entities, this filter removed any rules found in sentences where there weren’t rules generated corresponding to both a verb and a noun in the sentence. The idea was that these should generally be related, and if one is missing it likely indicates an incorrectly generated rule. Some of the examples this rule came from had the noun as the object, and others as the subject, so the rule so the filter was applied relatively broadly by just requiring a noun with the verb, rather than restricting the noun to a specific location or role in the sentence.

5.6.7 Filter Sentences with Negatives

While many rules are thought of as describing what a player can or can’t do, rules are generally introduced in a positive manner explaining what the player can do, and the player can’t take actions that aren’t described in the rulebook. As a result, many sentences with negative words like “can’t” were illustrative examples or elaborations and not the rules being introduced. This filter removed all rules proposed from sentences with these negative words.

5.6.8 Filter One-Keyword Sentences

Game rules are highly-interrelated, and the schema used represents rules as not just guidelines for actions but also the entities being acted upon, the players performing the actions, and other associated information. As a result, it's rare for there to be one rule not related to other rules that are nearby. This filter removed all rules which were the only rules proposed in a given sentence.

5.6.9 Filter Numeric Elaborations

Some rules are numeric in nature, but frequently when a sentence includes a large quantity of numbers it's walking through a particular scenario and providing an example to the reader. This filter removed rules proposed in sentences with a large quantity of numbers.

5.6.10 Verb Sentence Filter

Some verb forms indicate the likelihood that a sentence is elaborative rather than introducing a new rule. For example, most sentences introducing a mechanism for moving a piece use "move" and keyword matching is done at the lemma level to avoid minor morphological variations from reducing recall. However, it's common in elaboration text for the word "moving" to be used instead. This filter takes specific alternate forms of verb keywords used in elaborative sentences and excludes sentences containing them from generating rules.

5.6.11 Filter Sentences with Verb but not Noun Keywords

A variant on the rule discussed in section 5.6.6, this rule filters out sentences with a verb keyword which don't also have a noun keyword leading to another rule being extracted. The purpose was to be less aggressive at filtering than the other rule, allowing nouns that didn't have a verb counterpart but not allowing an action without the entity acting or being acted on.

5.6.12 Filter Sentences with Evaluative Words

When sentences evaluate a particular move, they are generally an elaboration to help the reader understand already-described rules or aid him in forming a strategy. These sentences are often revealed in the form of a word like "easily". Sentences with an evaluative keyword were excluded from generating rules by this filter.

5.6.13 Filter Ineffective Keywords with Multiple Rule Classes

While previously the filter described in section 5.6.3 was used to filter out ineffective keywords, there are a number of keywords that pertain to multiple classes of rules. By the standard relation extraction process, without needing any filters, many of these extraneous proposed rules are filtered out when they don't connect to other nearby rules, but in some cases they do form invalid connections and remain in the set of proposed rules. This rule focused on filtering out now just keywords that were ineffective, but rather for (keyword, rule class) pairs that were ineffective. The aforementioned filter described in section 5.6.3 handled keywords where none of the rules with the keywords were effective, so this further pruned keywords that were effective by further restricting the classes of rules proposed when there were multiple classes associated with the same keyword.

5.6.14 Apply Head Restrictions to Modifier Keywords

Some keywords representing rules are modifiers. For example, "your" may be differentiating the turn player's pawn from the other players' pawns. These modifiers can frequently be applied to a wide range of head nouns. The particular use of "your" mentioned above would be valuable when modifying "pawns", but would have an altogether different meaning when modifying "turn". This filter extended the modifier keywords to allow them to specify either the head noun keywords they should be associated with, or alternatively an exclusion list of head noun keywords they should not be associated with. By extending these keywords to also place restrictions on the head noun, when the modifiers were used in different contexts, this filter would prevent those rules from being extracted by the system.

5.6.15 Filter Attributes in Sentences without Rules

This filter does not affect the rules being extracted, but rather the attributes and attribute values (and, indirectly, relations). Since many rules were filtered out based on the other filters described in this section 5.6, this rule filtered out attributes extracted from sentences without rules also being extracted. This was an attempt to leverage the rule extraction filtering to improve the other entity extraction tasks, and indirectly the later relation extraction tasks.

5.7 Evaluation Methodology

The evaluation process involves using the system on a corpus of game rules and comparing each component of the system with a human-annotated gold standard.

5.7.1 Data

Four games have been annotated for the evaluation. Given the relative high time cost of human annotation, and given the assumption that there's a small set of mechanics that represent a wide array of games, games were chosen that have a similar rough structure: players have pieces which move along a track attempting to reach the end first. The choice of games was not affected by the text or structure of the rulebooks but rather the structure of the games.

A wider range of games could have been chosen to show broader applicability of the methods, but the focus was placed on illustrating that a common set of mechanics used in different ways can be learned even when the text and specific relationships between the mechanics vary. Nevertheless, there are still significant differences among the chosen games.

The four games chosen were: Candyland, Chutes and Ladders, Parcheesi, and Trouble. The rulebooks used were the rulebooks that come with a recent physical, commercial copy of each game, transcribed into a text file with casing and paragraph structure preserved.

The number of instances of each entity and relationship to learn for each of the games is listed in Table 9. This only includes those to be learned automatically, discussed further in Section 5.7.2.

The numbers in the table are repetitive and not coincidentally so. The number of attributes is equivalent to the number of rule-attribute mappings and attribute-value mappings since these relationships are one-to-one. The parent-child relationships to learn would also be redundant as one less than the number of rules since each rule has exactly one parent except for the root which has none, but the caveat already mentioned about only including those entities and relationships that are learned results in a smaller number for this subtask.

Table 9 - Instances to learn per subtask per game

Game	Rules	Attributes	Attribute Values	Parent-Child Relationships	Rule-Attribute Mappings	Attribute-Value Mappings
Chutes and Ladders	60	66	66	42	66	66
Candyland	74	90	90	56	90	90
Parcheesi	73	90	90	51	90	90
Trouble	56	71	71	35	71	71
Total	**263**	**317**	**317**	**184**	**317**	**317**

Overall, these four games include 897 entities and 818 relations to learn spread across the six subtasks. The ordering of children is not included in this table since this task is evaluated in a significantly different way than the other tasks.

5.7.2 Evaluation Process and Metrics

The evaluation process for the entity and relation extraction subtasks consists in comparing the system-generated entities and relationships to the human-annotated entities and relationships which are treated as a gold standard. For each subtask, precision and recall are used to evaluate the system's performance.

In order for an entity or relation to be considered correctly mapped, not only does the entity or relation need to appear in both the system's output and the manually-annotated rules but the text span representing the rule must at least overlap by one token. This is strict but prevents treating spuriously-generated rules as correct when they weren't generated correctly.

Additionally, only rules to be automatically learned are included in the evaluation. There are two classes of rules which are not learned automatically. Section 4 already described why some rules not described in text but indicated by physical objects in a game's box are not learned automatically.

There is another class of rules not yet described that also is not learned automatically. These are rules which are implicit either in all games or given other mapped rules. For example, every game has a Game rule, and each of its children are specified as having one instance. The attribute values are not specified, but the rules are implicit in the nature of games (as specified by the schema of mechanics) and thus do

not need annotation to be inferred. This also includes some other rules throughout the tree even children of automatically learned rules. For example, RepeatActionPerPlayer always has one child which is of the ActionSequence class. The children of ActionSequence are actions, and may vary in mechanic as well as quantity, but the intervening ActionSequence rule is not optional given the RepeatActionPerPlayer rule. As a result, all rules which are required by their parents are treated as not needing annotation and thus are not included in the evaluation.

When measuring the ordering of child rules, the only time ordering matters is when there's a variable passed between rules which needs to be mapped correctly. Of rule pairs where a variable is provided by one rule and required by another, the percentage that are in the correct order is used as a measure of ordering correctness. This is not a strict measure of ordering of all children.

5.7.3 Evaluating the Machine Learning-Based Entity Extractor

Since machine learning was used for comparison against the keyword-based entity extractor, the evaluation of the machine learning-based approach requires special attention. Since there is no large corpus of rules available for training on and only four rulebooks for evaluation, leave-one-out cross-validation was used at the rulebook level to evaluate the machine learning-based entity extraction.

Since there was a separate classifier for each entity class being extracted, cross-validation required one classifier for each entity class for each rulebook, with the training set being built from the three other rulebooks and the test set being built from the rulebook being tested on.

6 Results

The system's performance averaged across each of the entity and relation extraction tasks and overall is shown in Table 10.

Table 10 - Precision and recall by subtask groups

Task Group	Average Precision	Average Recall	Average F-Measure
Entity Extraction	6.9%	77%	13%
Relation Extraction	5.2%	55%	9.4%
Overall	**6.0%**	**66%**	**11%**

The system had a reasonably high recall while having a very low precision. The keyword approach resulted in many entities being identified, but many more entities were identified than were used in the rules. In the rulebook, many of these extra entities really were references to entities used in the game, but the references were providing ancillary information rather than describing a new rule. One example of this is shown in Listing 3.

Listing 3 - Example of entity references outside new rule introduction

> On your turn, spin the spinner and move your pawn, square by square, the number shown on the spinner. **For example, on your first turn, if you spin a 5, move to square #5 on the board.**

In this case, the first sentence describes a series of rules—the turn structure, an action taken on the turn (spinning the spinner), and the consequences of the action (moving a pawn). The following sentence also includes language about what happens on a particular turn, illustrating this aforementioned series of rules with an example. It describes a particular spin and its consequences relative to the pawn's starting position. Using a keyword-based approach, some similar rules are identified in each sentence. They're both treated as new rules, though, rather than rules and then illustrations of those rules.

The result is a large portion of the actual rules being identified, but also many more rule proposals which are not rules. The same applies to other entities and relationships between them.

The filtering that is applied to remove entities not well-connected in the rule hierarchy does not remove these entities because a number of entities are identified in the same descriptive text and linked together so they successfully make it through the filters.

While not all subtasks are equally effective, they all share this same pattern; it is not just one subtask dragging the overall numbers in this direction. The precision and recall numbers across all subtasks are shown in Table 11. The core Rules entity extraction task was the best for both precision and recall, but for every subtask the recall number was between seven and thirteen times higher than the corresponding precision number.

Table 11 - Precision and recall for all games

Subtask	Precision	Recall	F-Measure
Rules	9.1%	86%	16%
Attributes	6.4%	82%	12%
Attribute Values	5.1%	64%	9.5%
Parent-Child Relationships	5.6%	40%	10%
Rule-Attribute Mappings	4.8%	61%	8.9%
Attribute-Value Mappings	5.1%	64%	9.5%

The precision and recall by subtask is also shown for each individual game tested on. Table 12 shows the results for Candyland, Table 13 shows the results for Chutes and Ladders, Table 14 shows the results for Parcheesi, and Table 15 shows the results for Trouble.

Table 12 - Precision and recall for Candyland

Subtask	Precision	Recall	F-Measure
Rules	15.4%	88%	26%
Attributes	13.2%	91%	23%
Attribute Values	9.3%	64%	16%
Parent-Child Relationships	12.7%	41%	19%
Rule-Attribute Mappings	9.0%	62%	16%
Attribute-Value Mappings	9.3%	64%	16%

Table 13 - Precision and recall for Chutes and Ladders

Subtask	Precision	Recall	F-Measure
Rules	13.5%	88%	23%
Attributes	8.0%	86%	15%
Attribute Values	7.6%	82%	14%
Parent-Child Relationships	9.3%	43%	15%
Rule-Attribute Mappings	6.7%	73%	12%
Attribute-Value Mappings	7.6%	82%	14%

Table 14 - Precision and recall for Parcheesi

Subtask	Precision	Recall	F-Measure
Rules	5.9%	81%	11%
Attributes	4.2%	81%	8.0%
Attribute Values	3.0%	59%	5.7%
Parent-Child Relationships	2.9%	33%	5.3%
Rule-Attribute Mappings	3.1%	60%	5.9%
Attribute-Value Mappings	3.0%	59%	5.7%

Table 15 - Precision and recall for Trouble

Subtask	Precision	Recall	F-Measure
Rules	7.4%	86%	14%
Attributes	5.0%	68%	9.3%
Attribute Values	4.1%	55%	7.6%
Parent-Child Relationships	4.5%	46%	8.2%
Rule-Attribute Mappings	3.8%	51%	7.1%
Attribute-Value Mappings	4.1%	55%	7.6%

Some games seem to have better results than others, but not always in every category. One trend is clear, however: the system performs better on Candyland and Chutes and Ladders than Parcheesi and Trouble. This is quantified in Table 16.

Table 16 - Simple vs. complex games

	Simpler Games			Complex Games		
Subtask	**Precision**	**Recall**	**F-Measure**	**Precision**	**Recall**	**F-Measure**
Rules	14.5%	88.1%	24.9%	6.7%	83.3%	12.4%
Attributes	10.6%	88.8%	18.9%	4.6%	74.4%	8.7%
Attribute Values	8.5%	73.1%	15.2%	3.6%	56.9%	6.8%
Parent-Child Relationships	11.0%	42.0%	17.4%	3.7%	39.5%	6.8%
Rule-Attribute Mappings	7.9%	67.5%	14.1%	3.5%	55.4%	6.6%
Attribute-Value Mappings	8.5%	73.1%	15.2%	3.6%	56.9%	6.8%

Candyland and Chutes and Ladders are here considered simpler games than Parcheesi and Trouble. This is a qualitative statement, but evidenced by some significant differences in the gameplay. The more complicated games have each player moving multiple pieces and making choices, whereas the simpler

games only involve one piece to move per player and the outcome is dependent entirely on random factors without players making decisions. Candyland has at least as many rules to learn as Parcheesi for each subtask, so this is not an issue of the number of rules being greater but rather it is a statement about the complexity of the gameplay and the types of interactions between the rules.

This is difficult to quantify, but boardgamegeek.com users recommend that Parcheesi and Trouble be played on average by users ages 4.5 and above while the same age recommendation for Candyland and Chutes and Ladders is ages 3 and above (Boardgamegeek.com 2013).

The system performs better on both precision and recall for each subtask for the simpler games. Averaging across subtasks, the simpler games have a precision 5.9 percentage points higher and a recall 11 perentage points higher than the more complex games. These differences can be compared to the overall average precision of 7% and average recall of 67%. This indicates that even with similar numbers of rules and a similar schema, games that are qualitatively more difficult appear to be more difficult to automatically learn the rules to.

In addition to some games being more difficult to learn than others, some subtasks also have more variation than others. The system learned parent-child relationships with on average a precision of 7.4%, but the standard deviation of this number across games was 4.5%, from a low of 2.9% on Parcheesi to a high of 12.7% on Candyland. All of the subtasks had precision numbers with a standard deviation of 44-61% of the average. These numbers across all subtasks can be seen in Table 17.

Table 17 - Subtask precision and recall: variation across games

	Precision			Recall		
Subtask	**Average**	**σ**	**σ/Average**	**Average**	**σ**	**σ/Average**
Rules	11%	4.6%	44%	86%	3.4%	4.0%
Attributes	7.6%	4.1%	54%	82%	10%	12%
Attribute Values	6.0%	2.9%	49%	65%	12%	18%
Parent-Child Relationships	7.4%	4.5%	61%	41%	5.3%	13%
Rule-Attribute Mappings	5.7%	2.7%	48%	61%	9.0%	15%
Attribute-Value Mappings	6.0%	2.9%	49%	65%	12%	18%

The Ordering Children subtask was measured with a different metric than the precision and recall used for the other subtasks, so it has not been shown in the previous tables showing subtask performance. Given the heuristics used, the percentage of rule pairs with variables passed between them which were ordered correctly was 100%. Given the heuristics used, the only circumstances where this could be less than 100% involve having multiple producers of the same variable type followed by multiple consumers of the same variable type within an overlapping scope. These circumstances did not arise in any of the four games used for evaluation, so this was not a significant problem relative to the other subtasks for these games.

The relation extraction tasks take place after the entity extraction tasks. The poor precision of the entity extraction task makes the relation extraction tasks more difficult than they would be with better output from the entity extraction tasks. One way to measure the impact of this spillover in reduced effectiveness is by assuming all the correct entity extractions and using relation extraction on that clean set. The results of this experiment is shown in Table 18.

Table 18 - Relation extraction evaluated assuming perfect entity extraction

	System's Entity Extraction			Perfect Entity Extraction		
Subtask	**Precision**	**Recall**	**F-Measure**	**Precision**	**Recall**	**F-Measure**
Parent-Child Relationships	5.6%	40%	10%	83%	55%	66%
Rule-Attribute Mappings	4.8%	61%	8.9%	94%	86%	90%
Attribute-Value Mappings	5.1%	64%	9.4%	74%	69%	71%

Using the gold standard entity extraction, the relation extraction tasks improved dramatically. The precision increased from an average of 5.2% across the three subtasks to an average of 83%, which represents an increase by a factor of 16. The recall, which started off better, improved more modestly but still went up from 55% to 70%. F-Measure increased by a factor of 6.6.

To validate the assumption that keyword-based entity extraction would be superior to machine learning-based approaches given the small amount of data available for training. The results of this comparison are shown in Figure 7.

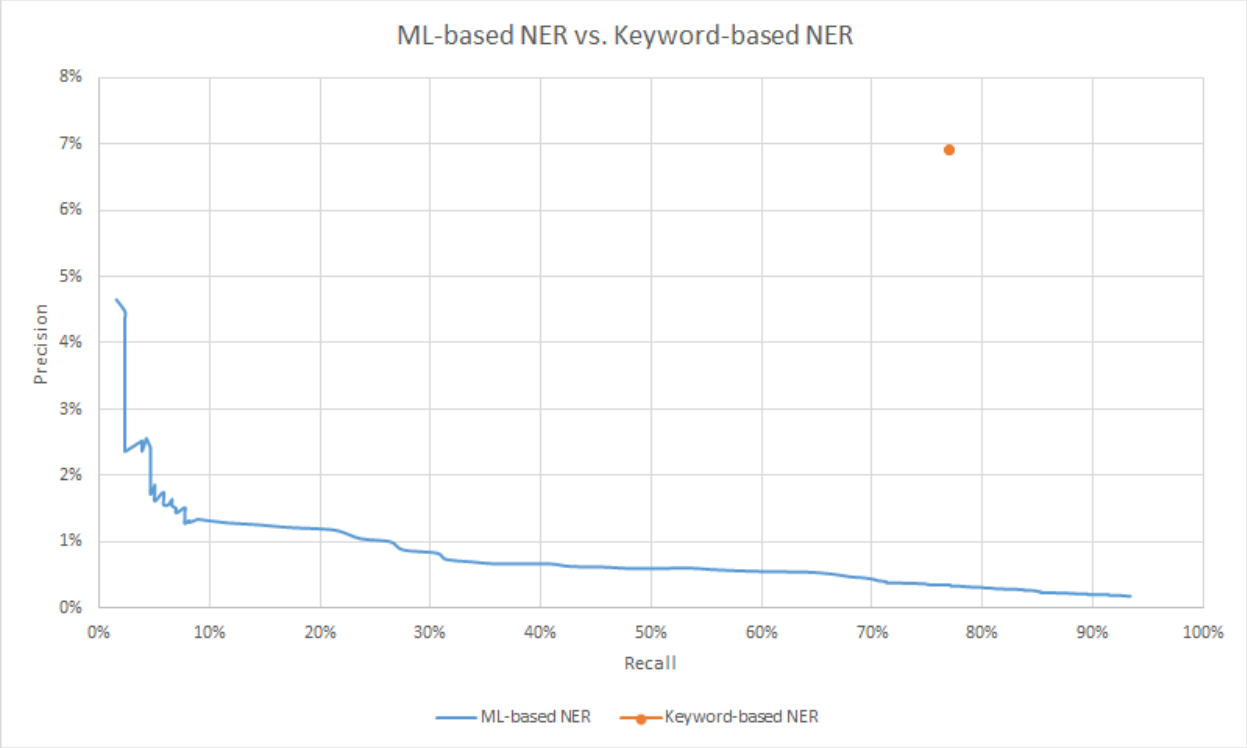


Figure 7 - Entity Extraction Comparison: Machine Learning and Keywords

While the keyword-based approach just has one operating point compared to the range of thresholds available with machine learning, the keyword-based approach performed clearly better, especially on precision which is where the keyword-based approach had its biggest challenges. The keyword-based approach had a reasonably-high recall but low precision, while the machine-learning based approach was not able to reach that precision at even a much lower recall, and with a 19.6 times better precision at the recall the keyword-based approach operates at.

This confirms that the problem is challenging and even classifier-based approaches get very low precision at reasonable levels of recall.

With the generic keyword-based approach outperforming the machine learning based approach, the additional filters described in section 5.6 were applied to the keyword system in an attempt to improve the system's precision. The results are shown in Table 19. Since many of the rules built on one another and the ordering is relevant, the filters are shown in the order they were experimented with. Rules that were

satisfactory are in green and those that were rejected are in red. Filters were applied on top of the best existing system at the time, so the filters represented by all green lines above it were included.

Table 19 - Filtering to Improve Rule Extraction Precision

Order	Experiment	Recall	Precision	F-Measure
1	Original	85.2%	9.1%	16.4%
2	Filtering Header Sentences	78.3%	9.1%	16.3%
3	Filtering Sentences with Too Many Rules	77.2%	9.3%	16.6%
4	Filter Ineffective Keywords	85.2%	10.4%	18.5%
5	Filter Conditional Sentences	85.2%	11.4%	20.1%
6	Filter Early Sentences	85.2%	11.8%	20.7%
7	Filter Sentences without both Verb and Noun Keywords	76.0%	13.7%	23.2%
8	Filter Sentences with Negatives	82.1%	16.5%	27.5%
9	Filter One-Keyword Sentences	82.1%	17.0%	28.2%
10	Filter Numeric Elaborations	80.6%	16.9%	27.9%
11	Verb Sentence Filter	82.1%	23.6%	36.7%
12	Filter Sentences with Verb but not Noun Keywords	79.5%	23.6%	36.4%
13	Filter Sentences with Evaluative Words	82.1%	24.1%	37.3%
14	Filter Ineffective Keywords with Multiple Rule Classes	76.4%	35.1%	48.1%
15	Apply Head Restrictions to Modifier Keywords	76.4%	36.0%	48.9%
16	Filter Attributes in Sentences without Rules	76.4%	36.0%	48.9%

Two-thirds of the filters turned out to work effectively. Since the rules were being proposed based on looking at the data from the first game, Chutes and Ladders, and then they were being generalized to the other three games, not all of the proposed rules generalized well. Fortunately, the ones that did led to a sizeable improvement, with precision going from 9.1% on the baseline keyword rule extraction system to 36% with all of the effective filters included. This came at a cost of 8.8 recall points, with recall dropping from 85.2% down to 76.4%.

The filtering of header sentences was ineffective in part due to the parser not tokenizing headers well and also due to different header structures for different games. Filtering sentences with too many rules ran into difficulties because even though it filtered out many rules that were incorrectly generated, there were good rules mixed in as well and recall dropped much more than precision rose. Filtering sentences without both verb and noun keywords also led to a dramatic drop in recall, and even just filtering out sentences with a verb but not a noun keyword was not enough to prevent the recall losses that made this

approach fail. Additionally, filtering numeric elaborations only had a very minor precision improvement and came at a cost to recall.

Filtering sentences with negatives, the verb sentence filter, and filtering ineffective keywords with multiple rule classes had the biggest positive impact, with the verb sentence filter being particularly effective in not causing a reduction in recall. These all worked as intended, and the overall effect of all the filters was a gain of about 3 points of precision for every point of recall lost.

The results of this filtering of rule extraction, the overall system bottleneck, had ripple effects that positively impacted the other extraction tasks as well. This impact is shown in Table 20.

Table 20 - Improved Overall Performance

Subtask	Original			Improved		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Rules	9.1%	85%	16%	36%	76%	49%
Attributes	6.4%	82%	12%	18%	75%	29%
Attribute Values	5.1%	64%	9%	14%	61%	23%
Parent-Child Relationships	5.7%	41%	10%	30%	37%	33%
Rule-Attribute Mappings	4.8%	62%	9%	14%	59%	23%
Attribute-Value Mappings	5.1%	64%	9%	14%	61%	23%

The other tasks, aside from rule extraction, went from an average of 5.4% precision and 62.6% recall to an average of 18.1% precision and 58.6% recall, an improvement by a factor of 2.3 in precision with similar to the rule extraction subtask a 3.1 point improvement in precision for each point of recall lost. In particular, the parent-child relationship subtask, the one most related to the rule extraction task, saw its precision improve by a factor of 4.3 with a 6.4 point increase in precision for each point of recall lost. The f-measure for the core rule extraction task increased by a factor of 3.1, from 16% to 49%, and the other subtasks improved by factors ranging from 2.4 to 3.3, with an average increase by a factor of 2.7.

The average f-measure across all subtasks was 30%, 2.8 times better than the pre-filter f-score of 11%.

7 Conclusion and Future Work

Humans are able to learn to play board games by reading a rulebook. Games have to fully explain the interactions between the various pieces, locations, and actions that can happen throughout the gameplay. A player's knowledge about the world does not allow him to infer additional rules based on recognizing the entities the various game pieces represent. Being self-contained in this way makes it a good domain for attempting to automatically learn a structured, interrelated set of knowledge from text. Some basic domain knowledge is assumed, such as an understanding about common game mechanics like rolling dice, but aside from that generic world knowledge has a much more restrictive role in rulebook text since the document needs to fully specify the logical system described therein.

A representation was proposed whereby a game's rules can be represented by a tree structure comprised of instances of a common set of mechanics. Four simple games were annotated in this manner, and given the small volume of data available a keyword-based entity and relation extraction system was built to extract the basic game entities (pieces, areas, actions) and relationships between those entities (actions, attributes, attribute values).

The keyword-based system involves proposing entities whenever keywords are encountered, greedily forming relationships between nearby proposed entities matching the schema of the nodes in the rule tree. Overall the system had a reasonably high recall of 77% for entity extraction subtasks. Given a large number of illustrative examples and elaboration in the rulebook, the system proposed many more entities than actually participate in the rules which yielded a low precision of 6.9%. A machine learning-based system was created for comparison and also showed very low precision, with the keyword-based approach clearly outperforming it due to the sparseness of training data available for the machine learning system.

Starting from this base of decent recall but low precision on entity extraction, relation extraction tasks added some additional error but maintained the same property of having a much higher recall than precision. The system overall had a recall of 66% with a precision of 6.0%. An experiment assuming perfect entity extraction revealed that relation extraction itself had a much better precision of 83%, 16 times more than the entity extraction task, with a slightly higher recall as well. This indicates that relation

extraction is performing relatively well while entity extraction is dragging the overall system performance down.

The keyword-based extraction system was then improved by applying a number of filters to remove rules extracted incorrectly, and this improved the precision of the system on entity extraction up to 22.7%, with the basic rule extraction notably rising to 36% precision. This was accompanied by a six point drop in recall across entity extraction tasks. The f-measure rose from 16% to 49% on the basic rule extraction, indicating the overall improvement was substantial even given the loss of recall.

The system worked better on simpler games than more complex ones, even when the simpler games had more rules than the complex ones. Identifying game mechanics had the best precision and recall of any of the entity extraction tasks for each game the rules were automatically learned for, and identifying attributes always led to a better performance than identifying attribute values.

Given that entity extraction proved to be the component dragging overall system performance down, even after filtering, future work will focus on improving that aspect of the system. This may include filtering out passages of rule text which have a repetitive structure relative to previous sentences or other filters.

Additional work will also follow-up on the impact of game complexity by annotating more complex games such as Settlers of Catan and providing a wider range of game complexity and a larger sample size of games to evaluate the impact of game complexity on system performance. More annotation may also result in large enough data sets to effectively apply more advanced entity and relation extraction techniques, such as those based on machine learning.

8 References

Elisabetih Andre, Kim Binsted, Kumiko Tanaka-Ishii, Sean Luke, Gerd Herzog and Thomas Rist. 2000. Three RoboCup simulation league commentator systems. In AI Magazine 21(1), pages 57-66.

Nguyen Bach and Sameer Badaskar. 2007. A Survey on Relation Extraction. http://www.cs.cmu.edu/~nbach/papers/A-survey-on-Relation-Extraction.pdf

Shumeet Baluja, Vibhu O. Mittal, and Rahul Sukthankar. 2000. Applying Machine Learning for High-Performance Named-Entity Extraction. In Computational Intelligence, Volume 16, Issue 4, pages 586-595.

Boardgamegeek.com. User Suggested Ages poll. http://www.boardgamegeek.com/boardgame/59451/candyland-sweet-celebration-game,

http://www.boardgamegeek.com/boardgame/5432/snakes-and-ladders,
http://www.boardgamegeek.com/boardgame/2136/pachisi, and
http://www.boardgamegeek.com/boardgame/1410/trouble. Accessed Jan 1, 2013.

S.R.K Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In Proceedings of ACL, pages 82–90.

S.R.K Branavan, Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: learning to map high-level instructions to commands. In Proceedings of ACL.

Jacob Eisenstein, James Clarke, Dan Goldwasser, and Dan Roth. 2009. Reading to learn: Constructing features from semantic abstracts. In Proceedings of EMNLP, pages 958-967.

Michael Fleischman and Deb Roy. 2005. Intentional context in situated natural language learning. In Proceedings of CoNLL, pages 104-111

David Kaiser. 2007. The structure of games. In ACM-SE 43: Proceedings of the 43rd annual Southeast regional conference, pages 61-62.

Tessa Lau, Clemens Drews and Jeffrey Nichols. 2009. Interpreting written how-to instructions. In IJCAI '09 Proceedings of the 21st international joint conference on Artificial Intelligence, pages 1433-1438.

Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza and Michael Genesereth. 2008. General Game Playing: Game Description Language Specification.
http://games.stanford.edu/language/spec/gdl_spec_2008_03.pdf

Raymond J. Mooney. 2008. Learning to connect language and perception. In Proceedings of AAAI, pages 1598-1601

David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. In Lingvisticae Investigationes, Volume 30, Number 1, pages 3-26.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In CoNLL '09 Proceedings of the Thirteenth Conference on Computational Natural Language Learning, pages 147-155.

Michael Thielscher. 2010. A General Game Description Language for Incomplete Information Games. In AAAI 2010.